A COMPUTATION-IMPLEMENTATION PARALLELIZATION APPROACH TO TIME-SENSITIVE APPLICATIONS

A Thesis
Presented to
The Academic Faculty

by

Bahar Çavdar

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
H. Milton Stewart School of Industrial and Systems Engineering

Georgia Institute of Technology
August 2014

A COMPUTATION-IMPLEMENTATION PARALLELIZATION APPROACH TO TIME-SENSITIVE APPLICATIONS

Approved by:

Professor Joel Sokol, Advisor
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor David Goldsman
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor William J. Cook
Combinatorics and Optimization
University of Waterloo

Professor Alejandro Toriello
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Professor Alan Erera
H. Milton Stewart School of Industrial and Systems Engineering
Georgia Institute of Technology

Date Approved: June 2014

To my beloved parents, Hacer and Cuma...

ACKNOWLEDGEMENTS

My most sincere gratitude goes to my advisor, Prof. Joel Sokol. His guidance and support made this thesis possible. I am grateful for all the time and effort he invested in me to cultivate my development as a researcher. His enthusiasm in research, valuable insights, commitment to his principles and positive attitude have been great motivational factors throughout my studies.

I want to thank Prof. Gary Parker for encouraging me to pursue my research interests. I am grateful for having the opportunity to work with Prof. Dave Goldsman. He was also very generous to have time to chat during my random visits. I would also like to thank Prof. Alan Erera. His comments helped me shape my research. I want to acknowledge Prof. Bill Cook for his guidance. He posed very interesting questions to open up new research problems. I also thank Prof. Alejandro Toriello for his time and effort to serve on my thesis committee and for his valuable suggestions. I am thankful to Prof. Craig Tovey for his very valuable feedback on my thesis. I would also like to extend my gratitude to Prof. Steve Hackman. He always shared his wisdom and invested so much to help me become a good teacher. I am also grateful to Prof. Chen Zhou for giving me the opportunity to teach. I would also like to express my gratitude to Prof. Meral Azizoğlu for believing in me and supporting me.

I am very thankful to Pam Morrison, Yvonne Smith and Mark Reese for their kindness and taking care of all the administrative work.

I have had so many great friends at Georgia Tech, who made my life in Atlanta the most memorable. I wish to thank ISyE gang, Can, Ezgi, Haiyue, Fangfang, Melih, Feng, Burak, Gustavo, Rodolfo, Soheil, Monica, Mallory, Vinod, Minkyoung, Evren, Helder, Aly, Alvaro, Andres, Cristobal, Guido, Daniel and Tim. I am thankful to

Didem, Şerife, and Tamara, and many others I should not have forgotten. I thank to all my friends for fruitful conversations we had, teaching me many things that I would have never learned somewhere else, all the fun I had during cultural dinners, concerts, road-trips, potlucks, vacations, playing games in the hall. They are the best gift Georgia Tech has given me.

My beloved friend Elçin was always there for me. She always cheered me up and encouraged me. I am also thankful to my great friends from METU for their encouragement over the years. I must also mention Hakan, Fatma and Tuba. They were very kind. Whenever I needed advice, they were always there.

I am very fortunate to have friends like Diego and Pratik, who dragged me out of my office to have fun. They were also great listeners, we shared so much and they changed my life in different ways.

I am especially grateful to İlke, Tuğçe, Murat and Oğuzhan for their irreplaceable friendship, love, and patience. They were my biggest supporters during the stressful times, I never felt alone. I do not remember how many times they listened to my presentations and gave feedback. I am deeply indebted to them for all the joy they brought to my life. Thanks to them, I had great memories in Atlanta.

I cannot express my gratitude enough for my roommate, officemate and best friend Işıl. She was always there to help me through difficult times and share my happiness. Nothing seemed difficult with her. We made it possible together, and she made Atlanta feel like home. Four years we spent at METU and five years at Georgia Tech, were full of fun, and it would not have been such a great journey without her.

Finally, I would like to send my deepest gratitude to my parents Hacer and Cuma, my brothers Erhan and Mehmet, my grandmother, uncles, aunts and cousins. I am eternally grateful to them for their unconditional love, and immense support to pursue my goals. I dedicate this thesis to the two most remarkable people in my life, to my parents.

Contents

List of Tables

List of Figures

SUMMARY

In this thesis, we study time-sensitive applications where it is important to minimize the completion time, i.e., time passing between receiving the instance and finishing the implementation of the solution. Different from the traditional approach, we are directly focusing on the minimization of the computation time as well as finding the optimal solution to the problem. The conventional approach to these conflicting objectives is generally to trade off one for the other. As an alternative, we propose a new approach called Computation-Implementation Parallelization (CIP), and develop methods to embed the computation time into the solution-implementation to minimize the total completion time. We implement our approach on specific cases of TSP and VRP. While working on the main approach, as a tangential study, we develop a distribution-free TSP tour length estimation model.

In the first chapter, we implement our CIP approach on a type of TSP we call the TSP Race problem. In this problem setting, given the same graph and starting from the same point, two salesmen are racing to finish the tour before the other. The race starts as soon as the graphs are released to the salesmen. Therefore, how long they compute is also a factor determining who will win. We demonstrate CIP's effectiveness on TSP Race instances. We also demonstrate a method for determining a priori when CIP will be effective. As an extension, we present a geometric analysis of the 2*opt* heuristic.

In the second chapter, we focus on large-scale parcel delivery. Making good routing decisions has become computationally more demanding due to both increasing size of the logistics operations and also the increasing trend in integrating the problems to make better decisions for the overall system. However, tight schedules in warehouses

may allow only very limited time for computation for routing. To improve the solution quality, increasing the computation time may not be always feasible since it requires either shifting the order cutoff time to an earlier point in time or delaying the truck dispatching. To overcome this issue, using our CIP approach, we embed computation into the truck loading phase to create additional time. As an extension to this chapter, we also propose a modification to the Granular Tabu Search by Toth and Vigo (2003), which improves solution quality by 2% on average on certain instances.

Our research in CIP opened up a new question on TSP tour length estimators. Tour length estimations can be used to determine the quality of a tour on hand. Estimation models in the TSP literature focus on instances where the node dispersion is known (and often is uniform). To deal with more general cases, i.e., graphs with non-uniform and unknown node dispersions, we develop a new empirical tour length estimation model for a broader dispersion family based on regression. Our model can estimate the tour lengths of these graphs to +/-3% of the tour length found by the Lin-Kernighan algorithm, with standard deviation of the estimation errors four times smaller than most previous models. When the distribution of the node coordinates is known, it can also estimate the coefficient of the well known asymptotic tour length estimation formula of Beardwood et al. (1959), whose estimate cannot be computed if the node dispersion is not known. Our model not only produces good estimates on several different dispersion types, but also substitutes for Beardwood et al.'s estimation formula when the distribution is unknown.

Chapter I

INTRODUCTION

In real time applications, the time spent finding a good solution and implementing it can be as important as the quality of the solution. For example, when we are executing a query on an unsorted data set, we want to find the shortest way of reaching all the requested data to minimize the user's waiting time. At the same time, we need to make this decision quickly, or else it will defeat the purpose of a fast solution. A conventional way of approaching this sort of problems is to use a quick heuristic solution method. In other words, solution quality is traded off with computation time.

In this thesis, we propose a new approach, a Computation-Implementation Parallelization (CIP) approach. The main mechanism of this approach is to embed the computation into the solution-implementation phase to minimize the total completion time, as an alternative to the standard compute-first implement-later approach. In this thesis, we specifically study time-sensitive spatial problems, and demonstrate the potential benefits of CIP.

In Chapter 2, we consider a problem we call the TSP Race, the case of two competing traveling salesmen. They are given the same graph, and the winner will be the one who comes back to the origin node first. The race starts just after the set of nodes is simultaneously released to both salesmen. Therefore, the winner is not necessarily the one who finds the shortest tour, because the time spent computing is also important. In this new setting, we implement CIP policies to embed some of the computation into the travel phase, i.e., computing later parts of the solution while implementing the earlier parts. We show that the CIP approach provides shorter completion times

in some cases. We also show how to derive a model to determine when and how CIP is likely to be better than the conventional compute-first implement-later methods, and we computationally show that the model is very accurate.

In Chapter 3, we implement the CIP approach on a class of multiple-vehicle routing problems. In distribution centers, once a set of orders is finalized, a computation is performed to determine how to route trucks to deliver the items. Then, the orders are loaded onto trucks according to this plan, and the trucks leave the warehouse. In warehouses that are serving a large number of customers, the tasks of finding optimized routes and of loading the trucks can both take a long time. In practice, time limitations are generally imposed on the computation time. If the operation schedule in a warehouse is very busy, decreased computation time can prevent finding good solutions for large-scale instances. An obvious suggestion to overcome this issue would be to increase the computation time. However, this requires either shifting the order cutoff time to an earlier time or delaying the truck dispatching, both of which decreases customer services. To overcome this issue, we implement our CIP approach to create more time for computation without disturbing the other operations in the warehouse. Using CIP policies, we embed the computation time into the loading time. We name this new problem setting the Computation-Time Limited Vehicle Loading and Routing Problem (CTL-VLRP). Implementing CIP enables us to find high quality solutions.

In addition to CIP, we report in this thesis two other discoveries we made in the process of our CIP work.

In Chapter 3, we also propose a modification to Toth and Vigo's [67] Granular Tabu Search (GTS) algorithm. In order to reduce the computation time, Toth and Vigo discard the shorts arcs, i.e., arcs whose lengths are shorter than a threshold defined in their paper, and perform tabu search on the new graph. The goal is to eliminate the non-promising moves from the neighborhood so that each iteration can

be performed faster. We show that discarding arcs if the polar angle between two nodes is larger than some threshold provides a 2% improvement on uniform random instances with large number of nodes.

Finally, in Chapter 4, we find a new distribution-free TSP tour length estimation model. Tour length estimation models are useful especially when we only need to know the approximate length of the optimal tour rather than the exact length, such as when solving the Location-Routing Problem (LRP). Some heuristics developed for this integrated problem of facility location and vehicle routing iteratively approximate the length of TSP tours, and use this information in the facility location phase. On randomly generated graphs, our model performs as well as a classic result in the literature, Beardwood et al.'s [7]. However, whereas Beardwood et al.'s model needs the dispersion of the nodes to be known, our model is distribution-free and uses only the node coordinates.

Chapter II

TSP RACE: MINIMIZING COMPLETION TIME IN TIME - SENSITIVE APPLICATIONS

2.1 Introduction to TSP Race

The classical traveling salesman problem (TSP) is to find the shortest tour through a set of points and back to the start; it is named after the situation of a traveling salesman who needs to visit a set of cities and then return home.

Consider a case where two traveling salesmen are given the same set of points and are each challenged to finish a tour before the other. The first traveling salesman follows the traditional approach, computing a tour upfront before starting to travel. The second one, on the other hand, makes a quick start. Each time he travels to a node, he simultaneously computes the next node to visit while traveling. One could imagine this traveling salesman driving from city to city with a laptop open on the passenger seat calculating where to go next. The second traveling salesman can be expected to travel longer due to the less global nature of his decision-making; however he has the advantage of not investing much time in computing beforehand. Because of this trade-off, it is not a trivial task to determine which traveling salesman will be the winner. In this chapter, we introduce an approach to this problem exemplified by the second traveling salesman.

We refer to this setting of TSP, with the objective of minimizing total completion time, as TSP Race. The statement of TSP Race is as follows: Given an instance of TSP, finish visiting all nodes and return to start as soon as possible.

The idea of CIP is generalizable to other problems where the computation and the implementation need not occur sequentially. Most of the current literature focuses on

minimizing the implementation time, but if minimizing the time between receipt of the problem and completion of the solution is important and the computation time is comparable to the implementation time, conventional methods are not necessarily efficient. The reason is that the implementation resources stay idle during the computation phase, and vice versa.

The rest of this chapter is organized as follows. In Section 2.2, we discuss the TSP applications that are compatible with the TSP Race setting. We are unaware of any previous work on TSP Race, but in Section 2.3, we present a brief review of general solution approaches to TSP. In Section 2.4, a general procedure of computation-implementation parallelization is introduced. In Section 2.5, we discuss methods to partition the graph and present computational comparisons. Section 2.6 shows computational results demonstrating the effectiveness of our approach. In Section 2.7, we develop a parameter selection mechanism for the parallelization approach. In Section 2.8, we present some final remarks.

2.2 Application Areas and Motivation

Our main motivation to design CIP approaches is the time-sensitive applications in which computation time is comparable to the implementation time, and thus the objective is to minimize the total completion time. Due to advancing technological improvements time-sensitive applications have been emerging in different areas. One example is scheduling of large-scale jobs to parallel processors in areas such as telecommunications, medical imaging, etc. [27]. Minimizing the completion time in multiprocessor systems can help to increase the utilization of the workstations. Arc therapy in cancer treatment is another time-sensitive real-time application. In some specific methods, computation time constitutes a significant portion of the total completion time [71, 61, 13, 65]. CIP approach can provide benefits in radiation therapy to minimize the total time a patient spends during the operation, i.e., computing the

solution and implementing the treatment.

Contemporary applications of TSP, which are our main focus in this chapter, can also be classified as time-sensitive applications. *The Traveling Salesman Problem: A Computational Study* by Applegate, Bixby, Chvătal and Cook [1] gives an elaborate list of applications for TSP. When there are large number of nodes over a small area, TSP Race concept might be more suitable than the traditional TSP. The examples are manufacturing (customized computer chips, drilling circuit boards, soldering printed circuit boards), data management and lasers and X-ray applications (crystallography, manufacturing of crystal art with a laser), etc. Since, many of the instances in such applications are not repeating, e.g., a query executed to extract large amount of data from an unsorted data set, following the traditional way of compute-first implement-later may not be efficient. Instead, minimizing the total completion time increases the throughput rate in systems where the product is highly customized.

2.3 Literature Review

TSP has a rich literature. [1] is a comprehensive guide to TSP's history. Exact TSP algorithms include branch-and-bound, e.g., [41, 42, 40, 69, 34, 14] and, cutting plane methods, e.g., [38, 3].

For most large TSP applications, exact algorithms may not be computationally feasible, even more so when the computation time is a part of the objective. Many heuristic algorithms have been developed over the years, to sacrifice travel time for decreased computation time. These heuristics can be grouped into construction, local optimization, and metaheuristics [48]. High quality solutions can be obtained in seconds by local optimization, simulated annealing and genetic algorithms; see, for example [23, 8, 54, 49, 56]. In this chapter, we focus on local optimization methods. For very large graphs, these heuristics are preferred to exact methods due to computational speed. However, their mechanism is the same as the exact algorithms, i.e.,

the computation is completed prior to starting to travel.

To our knowledge, the exact question of balancing computation and travel time by computing while traveling on a known set of nodes has not been studied in the literature. One related approach is to use online algorithms. However, online algorithms use all the information available. For example, [2] recomputes the tour whenever new information is available. Since we begin with complete information about the graph, an online algorithm will behave the same as the compute-first implement-later approach. In this chapter, we introduce a new approach aiming at minimizing the total completion time. To show its effectiveness, we demonstrate its use with the 2*opt* [23] and nearest neighbor (NN) [47] heuristics.

2.4 Theoretical Approach

Our approach is to decrease the total completion time by making better use of the travel time. The main idea is to embed the computation into travel so as to have less *computation-only* time. Figure 1 is a visual summary of our approach. We split the graph into subgraphs. Initially, we do one step of computation to choose a path through the first subgraph. While traveling on this path, we do a second computation to choose a path through the second subgraph, etc. Except for the first computation and the last travel, at each step a computation phase is embedded into a travel phase.

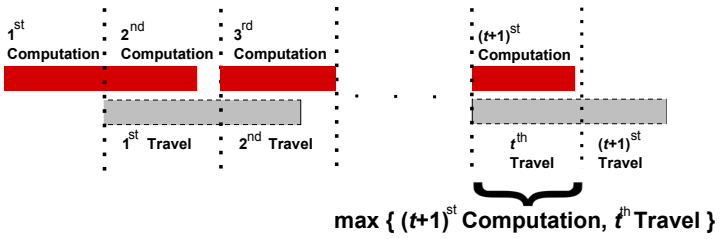


Figure 1: Travel-Computation Parallelization.

At each intermediate step, the elapsed time is the maximum of the travel time

and the computation time. As shown in Figure 1, since the second computation can be finished while traveling on the first subgraph, the completion time of that portion is equal to the first travel time. On the other hand, when we finish traveling on the second subgraph, we have to wait for the end of the computation to determine the next node to go to. So, the completion time of that portion is equal to the third computation time.

Before discussing the details of the approach, we first introduce some notation. While most of the notation is more general, for the remainder of this chapter we assume a complete Euclidean graph on a plane.

Notation:

- n: number of nodes in the graph,
- $t+1$: number of subgraphs,
- $T_i^{\mathcal{A}}$ ($T^{\mathcal{A}}$): travel time on the i^{th} subgraph (entire graph) using algorithm $\mathcal{A}$'s solution,
- $C_i^{\mathcal{A}}$ ($C^{\mathcal{A}}$): computation time of algorithm $\mathcal{A}$ to find the path on the i^{th} subgraph (entire graph),
- c: coefficient to express the ratio between the computation and the travel speed,
- h (w): length (width) of rectangular area containing the nodes.

Instead of running algorithm $\mathcal{A}$ on the entire graph and having the total completion time of $C^{\mathcal{A}} + T^{\mathcal{A}}$, we split the graph into $t+1$ subgraphs and run a Travel-Computation Parallelized (CIP) algorithm, embedding t computation steps into travel as in Figure 1. In order to find the first path quickly, we use a simple and fast algorithm $\mathcal{B}$, which would not be preferred for the entire graph due to its inferior average performance. For the subsequent subgraphs, $\mathcal{A}$ is run while traveling on the previous

subgraph's predetermined path. Since we interfere with the usual mechanism of the base algorithm $\mathcal{A}$ by not letting it optimize over the entire original graph G, we expect the total travel time to be longer. On the other hand, *computation-only* time can be reduced, possibly even to just the initial humble computation required by algorithm $\mathcal{B}$ to find the first path. That means the CIP algorithm creates a trade-off between the two contributors of the objective, i.e., the travel time and the computation time. For the gain in the computation-only time to outweigh the increase in the travel time, the subgraph partitioning for the CIP approach should satisfy the following inequality:

$$C^{\mathcal{A}} + T^{\mathcal{A}} \geq C_1^{\mathcal{B}} + \max\{T_1^{\mathcal{B}}, C_2^{\mathcal{A}}\} + \sum_{i=2}^{t} \max\{T_i^{\mathcal{A}}, C_{i+1}^{\mathcal{A}}\} + T_{t+1}^{\mathcal{A}} \tag{1}$$

The expression on the left is the total completion time when the main algorithm $\mathcal{A}$ is run on the entire graph. On the right is the completion time of the CIP approach. In practice, of course, each C and T will not be known exactly, and must be estimated.

In the rest of this chapter, we will use the 2*opt* algorithm as the main algorithm $\mathcal{A}$ and NN as the humble startup algorithm $\mathcal{B}$. We refer to this setup as $\text{CIP}_{\text{NN}}^{2opt}$. 2*opt* is one of the most commonly used tour improvement heuristics, and computational studies show that 2*opt* can find a tour which is in 6.4% of the Held-Karp bound on average [48]. However, the proposed approach can be implemented using any TSP algorithms, i.e, $\text{CIP}_{\mathcal{B}}^{\mathcal{A}}$, with minor modifications.

2.5 Partitioning the Graph

The main contribution of the CIP approach is to decrease the total completion time in spite of a possible increase in both computation time and travel time. How we partition the graph plays an important role. We propose and test four different partitioning rules. In the first three partitioning methods, the nodes are partitioned based on their location, before any travel begins. In the last method, the partition is created on the fly by the initial fast algorithm. These methods are discussed in detail

below.

1. **α-box** Draw a box with dimensions $h\sqrt{\alpha}$ by $v\sqrt{\alpha}$ in the corner of the graph, as shown in Figure 2. NN constructs a path through all the nodes in the α-box. While traveling on this path, NN constructs a tour on the nodes outside the box, and 2*opt* is run to improve the tour. When 2*opt* is finished and all the nodes in the box have been visited, travel continues on the remaining nodes using the sequence formed by 2*opt*.

2. **α-strip** This method is similar to α-box method except that the first path is constructed on the nodes falling in the strip whose length is $h\alpha$, as depicted in Figure 3.

3. **Matrix-Partitioning** We divide the rectangular region by a grid into equal-dimension subgraphs. NN is run on the first subgraph to construct a path. On other subgraphs 2*opt* is run, and the subgraphs are visited in order of an optimal tour of the centers of the grid squares. Figures $4(a)$-$4(c)$ depict cases where we divide the graph into 4, 16, and 36 equal-area subgraphs respectively. Matrix-partitioning can provide a greater saving in the computation-only time, because we are not only embedding the computation into travel, but also decreasing the amount of computation done by each 2*opt* execution. However, the travel time can get larger.

4. **Free-NN** In this method, the graph is divided into two subgraphs. Starting from the initial node s, NN is run to determine the next αn nodes to visit. While we are traveling on that path, a tour on the rest of the nodes is constructed by NN and improved by 2*opt*.

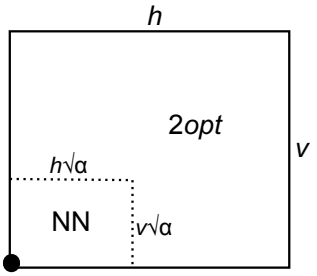


Figure 2: α-box method.

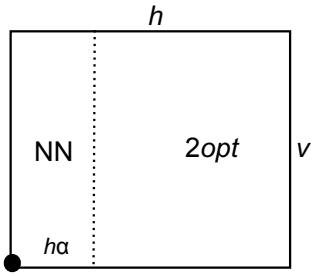
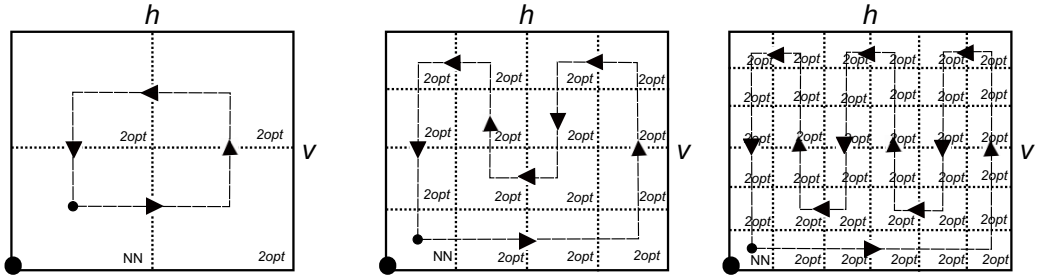


Figure 3: α-strip method.



(a) 4 subgraphs. (b) 16 subgraphs. (c) 32 subgraphs.

Figure 4: Matrix-partitioning.

The α-box, α-strip and free-NN methods divide the graph into two subgraphs, so we call them two-partitioning methods.

Having proposed four methods to partition the subgraphs, we performed computational experiments to compare the behavior of total completion time in different scenarios. In those experiments,

- the graphs are rectangular and the nodes are uniformly randomly distributed,
- c is assumed to be 1, and
- it is assumed that 1 unit of distance is traveled per second.

We tested the methods for different values of α, and performed computational experiments for different numbers of nodes, areas, and aspect ratios of the rectangles. For brevity, we report only the results on 10,000-node graphs for the four partitioning

methods on $1unit^2$ square graphs (Figures 5-8) and $100unit^2$ square graphs (Figures 9-12), since the behavior is similar in all cases.

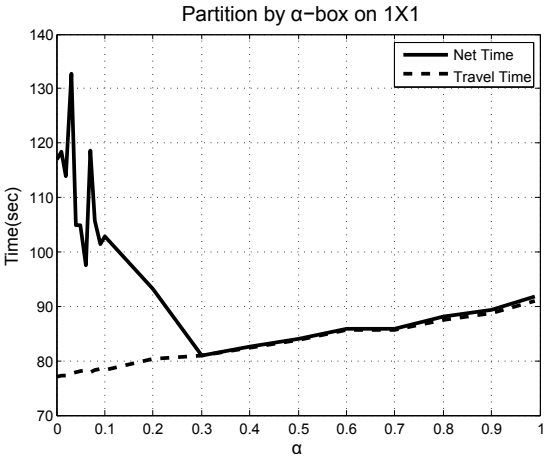


Figure 5: Completion time when partitioning is by α-box method.

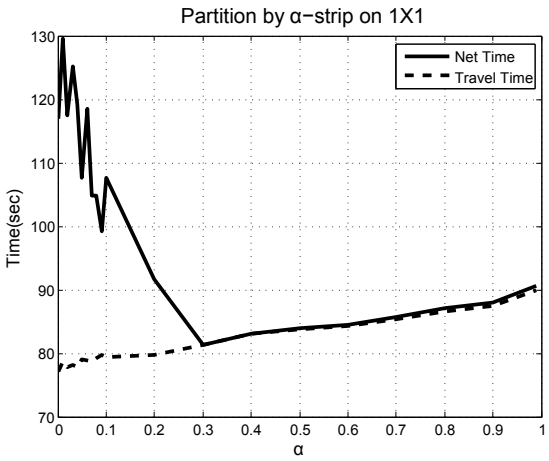


Figure 6: Completion time when partitioning is by α-strip method.

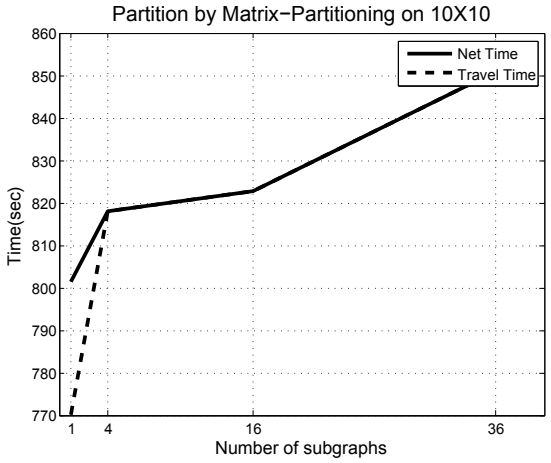


Figure 7: Completion time when partitioning is by matrix-partitioning.

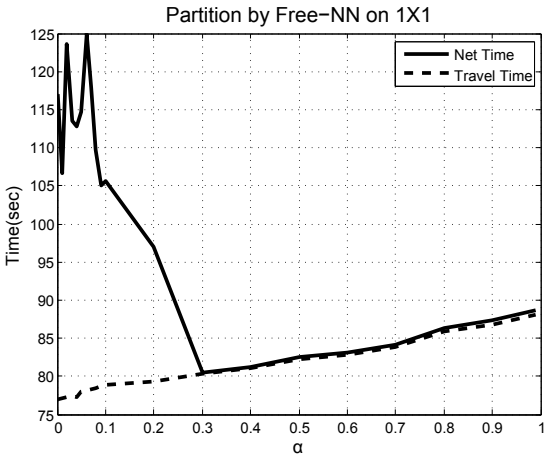


Figure 8: Completion time when partitioning is by free-NN method.

In the figures, we show net completion time (NT) and travel time (TT). The difference between NT and TT is the computation-only time (CT), the computation that could not be embedded into travel. In the two-partitioning methods (Figures 5, 6, 8) TT is increasing in α. With increasing α, NN determines a bigger portion

of the tour, preventing 2*opt* from making as many improvements. On the other hand, CT is generally decreasing to the initial NN time with increasing α, because of the embedding of computation into travel. At some value of α, NT nearly merges with TT, because the length of the first path includes all the computation required to find the path on the second subgraph. As long as NT at a specific α is less than the NT when α=0, which corresponds to running 2*opt* on the entire graph, parallelizing computation with travel performs better. For matrix-partitioning, we observe a similar pattern in NT, CT, and TT.

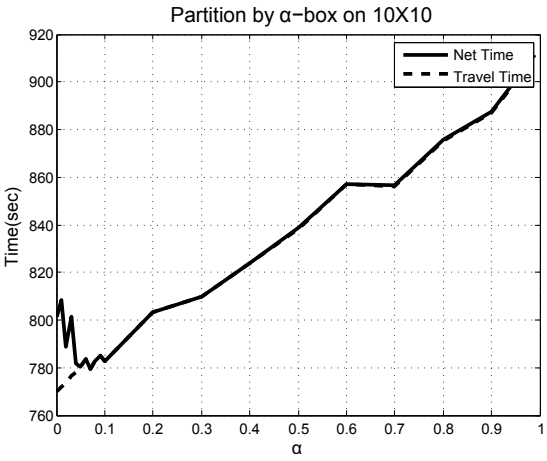


Figure 9: Completion time when partitioning is by α-box method.

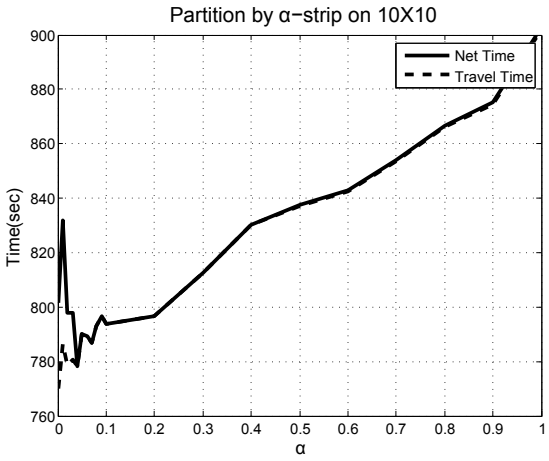


Figure 10: Completion time when partitioning is by α-strip method.

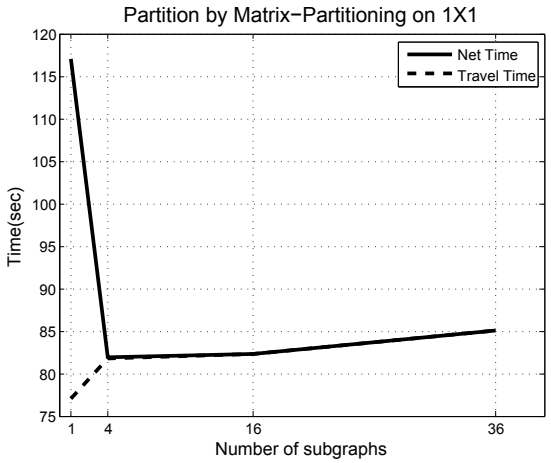


Figure 11: Completion time when partitioning is by matrix-partitioning.

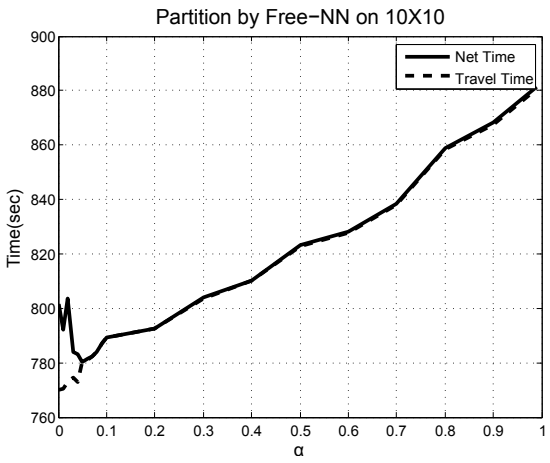


Figure 12: Completion time when partitioning is by free-NN method.

Figures 9-12 display the results of the same experimental setting for the same node set, but the area of the graph is enlarged by a factor of 100. NT, CT, and TT have similar patterns as before, but NT and TT merge earlier because travel time increases more quickly.

2.6 Computational Results

When parallelizing the 2*opt* algorithm with NN, the key question is whether the CIP approach reduces the total completion time. Below we present computational results showing the reduction in total completion time achieved by using $\text{CIP}_{\text{NN}}^{2opt}$. We use straightforward full implementation of NN and 2*opt* coded in C, and run on a heterogeneous cluster of machines with Xeon E5645 processor or near equivalent.

In Table 1, we compare $\text{CIP}_{\text{NN}}^{2opt}$ using each partitioning mechanism with running 2*opt* or NN on the entire graph. The numbers in the table show completion time scaled to where 2*opt* takes 1 second. For example, the completion time is 0.77sec if we run $\text{CIP}_{\text{NN}}^{2opt}$ instead of 2*opt* (1sec) on the 15,000-node, $16unit^2$ graph. None of the graph partitioning methods purely dominates the others, though free-NN is slightly better on average than other partitioning methods. More importantly, these

computational results verify that there are settings in which our CIP method can achieve significant improvement in total completion time compared to its component methods.

Table 1: Comparison of $\text{CIP}_{\text{NN}}^{2opt}$ to 2*opt* and NN in terms of the scaled completion time. The numbers in the table are the completion times scaled so that 2*opt* is 1.

		$\text{CIP}_{\text{NN}}^{2opt}$ Methods				Component Methods	
A	n	Free-NN	α-box	α-strip	Matrix-Partitioning	NN	2*opt*
0.25	5K	0.83	0.82	0.83	0.84	0.91	1.00
	8K	0.62	0.63	0.63	0.62	0.69	1.00
	10K	0.58	0.59	0.59	0.59	0.64	1.00
	15K	0.32	0.32	0.32	0.31	0.35	1.00
1	5K	0.89	0.89	0.89	0.93	1.00	1.00
	8K	0.74	0.75	0.76	0.77	0.84	1.00
	10K	0.69	0.69	0.69	0.70	0.76	1.00
	15K	0.47	0.48	0.48	0.46	0.52	1.00
16	5K	0.97	1.11	0.99	1.04	1.11	1.00
	8K	0.92	0.93	0.94	0.97	1.06	1.00
	10K	0.92	0.91	0.92	0.95	1.02	1.00
	15K	0.77	0.79	0.78	0.79	0.88	1.00
64	5K	0.98	0.98	1.01	1.05	1.13	1.00
	8K	0.96	0.97	0.97	1.02	1.11	1.00
	10K	0.96	0.95	0.96	1.00	1.08	1.00
	15K	0.87	0.87	0.89	0.90	1.00	1.00
100	5K	0.98	0.98	1.01	1.05	1.13	1.00
	8K	0.97	0.98	0.98	1.03	1.12	1.00
	10K	0.97	0.97	0.97	1.02	1.10	1.00
	15K	0.91	0.91	0.92	0.95	1.04	1.00
256	5K	0.99	0.99	1.01	1.06	1.14	1.00
	8K	0.98	0.99	0.99	1.04	1.13	1.00
	10K	0.97	0.98	0.98	1.03	1.11	1.00
	15K	0.94	0.94	0.95	0.98	1.08	1.00

As expected, NN can beat 2*opt* in small graphs with a large number of nodes, since in such cases the weigth of the computation time of 2*opt* in the completion time is more. As the graph area increases, NN loses its advantage of short computation

time due to worse tour quality. However, we see that for each setting it is possible for $\text{CIP}_{\text{NN}}^{2opt}$ to beat both NN and 2*opt* in terms of completion time. Gains in the completion time using the $\text{CIP}_{\text{NN}}^{2opt}$ are significant when there is a large number of nodes on a small area.

One might also wonder how $\text{CIP}_{\text{NN}}^{2opt}$ compares to a high-quality TSP algorithm. On the same set of instances as in Table 1, we compared $\text{CIP}_{\text{NN}}^{2opt}$ with Applegate et al.'s [44] implementation of the Lin-Kernighan (LK) algorithm [54]. Even with our naive implementation of NN and 2*opt*, $\text{CIP}_{\text{NN}}^{2opt}$ was up to 15% faster than LK on instances with smaller areas. On instances with larger areas LK was up to 7% better than $\text{CIP}_{\text{NN}}^{2opt}$, and in the sparsest cases LK would beat $\text{CIP}_{\text{NN}}^{2opt}$ even if we could entirely eliminate computation time from our algorithm. The bottom line is that for sufficiently dense instances a CIP approach using naively-coded simple heuristic can beat a cleverly coded high-quality heuristics, and for sufficiently sparse instances the opposite is true. In the next section, we discuss how to find break-even points between approaches.

Inequality (2) demonstrates the possible trade-off between the decreasing computation - only time and the increasing travel time due to preventing 2*opt* from making some improvements. However, we may not always face this trade-off. Due to its myopic nature, at the end of a tour NN sometimes needs to use one or more long arcs that are not part of the global optimal or a local optimal solution. Therefore, the first arcs added to the tour are less likely to be swapped by 2*opt* compared to the later ones. In fact, there could be cases in which 2*opt* would not swap any of the edges in the first path. That means the full quality of 2*opt* could be achieved using only the $C_{G_1}^{\text{NN}}$ extra computation time. Such cases are unlikely, but in Appendix 2.9 we analyze the necessary conditions for this to happen.

2.7 Time Coefficient Break-Even Analysis and Parameter Selection Mechanism

Now that we have shown the CIP approach to be effective, we would like to characterize the break-even point beyond which the $\text{CIP}_{\mathcal{B}}^{\mathcal{A}}$ approach should be used instead of its constituent algorithms $\mathcal{A}$ and $\mathcal{B}$. The main factor affecting the performance of the CIP approach is the ratio of computation and travel speeds. In applications where the computation is already extremely fast relative to the travel, the CIP approach may not help. So, we would like to determine the smallest applicable time coefficient c, i.e., the fastest computation or the slowest travel, for which the CIP approach will decrease the total completion time. We first find the break-even c in terms of the other parameters, and then develop an estimation function so that we can determine which value of α should be used in different settings. Of course, this analysis depends not only on travel speed and processor speed, but also on the algorithms used and the efficiency of their specific implementations. We perform the analysis for the free-NN method on $\text{CIP}_{\text{NN}}^{2opt}$; for other partitioning methods and algorithms corresponding results can be obtained following the same procedure.

The following is the restatement of inequality (1) for $\text{CIP}_{\text{NN}}^{2opt}$.

$$cC^{\text{NN}} + cC^{2opt} + T^{2opt} \geq cC_{G_1}^{\text{NN}} + \max\{c(C_{G_2}^{\text{NN}} + C_{G_2}^{2opt}), T_{G_1}^{2opt}\} + T_{G_2}^{2opt}. \qquad (2)$$

We need $c \in \left[\frac{T_{G_1}^{2opt}+T_{G_2}^{2opt}-T^{2opt}}{C^{\text{NN}}+C^{2opt}-C_{G_1}^{\text{NN}}}, \frac{T_{G_1}^{2opt}}{C_{G_2}^{\text{NN}}+C_{G_2}^{2opt}}\right]$ to have less completion time than $2opt$ and no computation-only time except for the initial computation performed by NN. We define the break-even c as the fastest computation or the slowest travel environment for which computation-implementation parallelization is likely to be helpful, and this corresponds to the lower bound of the interval.

Of course, none of $C^{\mathcal{A}}$, $C_G^{\mathcal{A}}$, $T^{\mathcal{A}}$ and $T_G^{\mathcal{A}}$ are known, so we run computational tests to observe the behavior of c in practice. Employing the same experimental design

as in Section 2.5, we performed another set of computational studies to observe the break-even c for different parameters. Results on unit square graphs with 5,000, 8,000 and 10,000 nodes are demonstrated in Figure 13. The first plot shows the break-even c values for different values of α. As we can see, having more nodes in a graph gives more flexibility to the CIP approach in terms of the applicability. More specifically, when there are more nodes in the same area, CIP is more effective even if the computation is very fast or travel is slow. The second plot is another interpretation of the break-even c. It shows us how much we can deviate from the existing setting of our computational experiments and still do better in terms of the completion time. In a specific TSP application where the travel is ρ times slower than what we assume in the computational experiments, and the computation is ω times faster, as long as $\rho\omega$ is less than the tolerance factor in Figure 13, $\text{CIP}_{\text{NN}}^{2opt}$ is still better than running $2opt$ itself.

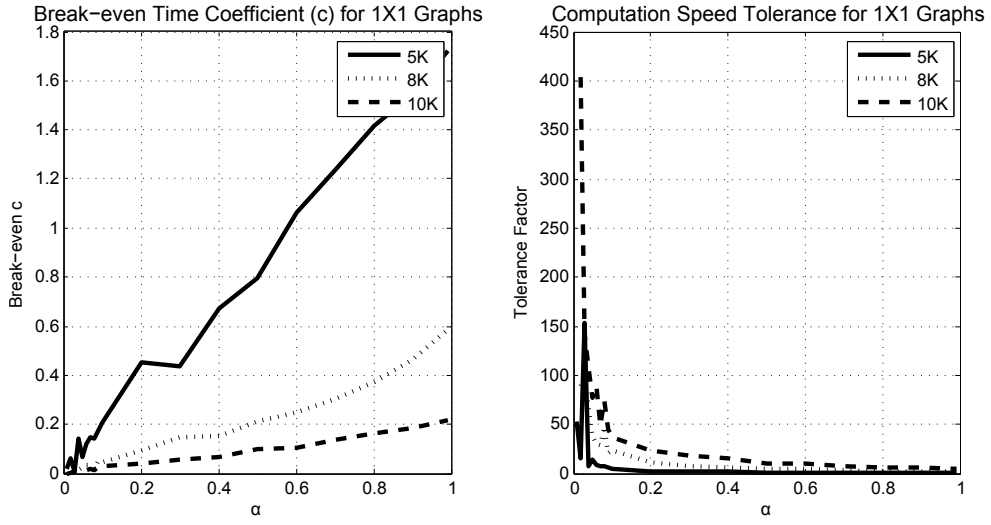


Figure 13: Break-even time coefficient and tolerance to change in computation or travel speed with respect to different values of α for 5,000, 8,000 and 10,000 nodes.

The computational results are important to help us to develop a notion of the interaction between the break-even c and the influential factors. Prior to setting up

the estimation model, we observed that break-even c

- increases in α,
- decreases in n,
- increases in the area of the graph,
- increases in the aspect ratio, v/h.

Guided by preliminary observations and the computational results, we used regression to find the following model to estimate the value of break-even time coefficient.

$$\left(\frac{c}{\sqrt{A}}\right)^{0.12} \approx x_1\alpha + x_2\alpha^2 + x_3(\alpha - 0.5)^2 + x_4 n + x_5 n^2 + x_6 n^3 + x_7\frac{v}{h} + x_8\frac{v}{h}\alpha \quad (3)$$

The data set used to train estimation model (3) comes from a wide range pool. The parameters are the combinations of $(n, \alpha, A, \frac{v}{h})$ generated from the following individual sets.

- $n \in \{10K, 12K, \ldots, 26K\}$,
- $\alpha \in \{0.01, 0.02, \ldots, 0.1, 0.2, \ldots, 0.9, 0.99\}$,
- $A \in \{1, 16, 49, 64, 100, 196, 400, 900\}$,
- $\frac{v}{h} \in \{1, 4, 16, 64\}$.

The coefficients x of the estimation model for the break-even time coefficient c in equation (3) are shown in Table 2.7.

Table 2: Coefficients of the model in equation (3).

	$\boldsymbol{\alpha}$	$\boldsymbol{\alpha^2}$	$\boldsymbol{(\alpha - 0.5)^2}$	$\boldsymbol{n}$
Estimate	3.94e+00	-3.71e+00	3.41e+00	-3.90e-05
Std. Error	7.34e-02	7.34e-02	7.34e-02	3.37e-06
	$\boldsymbol{n^2}$	$\boldsymbol{n^3}$	$\boldsymbol{\frac{v}{h}}$	$\boldsymbol{\frac{v}{h}\alpha}$
Estimate	1.39e-09	1.92e-14	1.09e-04	5.28e-04
Std. Error	1.96e-10	3.65e-15	1.88e-05	3.99e-05

The model's R^2 is 0.9987, and all the predictors are significant.

To validate the model we tested the estimation function on out-of-sample data and graphs different with sizes and areas. The parameters of the test set are combined with the following sets:

- $n \in \{10K, 12K, \ldots, 18K\}$,
- $\alpha \in \{0.01, 0.02, \ldots, 0.1, 0.2, \ldots, 0.9\}$,
- $A \in \{1, 4, 16, 64, 324, 1024\}$,
- $\frac{v}{h} \in \{1, 4\}$.

The testing procedure has two stages. In the first stage, we estimate the time coefficient c by equation (3). Then, this coefficient is used in computational experiments. For the same time coefficient c, we run both $\text{CIP}_{\text{NN}}^{2opt}$ (using the corresponding α) and $2opt$ on the same graph. We call these two runs *conjugate* runs. As it is implied by the definition of the break-even time coefficient, the NT's of these runs should be approximately equal. In our test results, we observe that the ratios between the NT's of the conjugate runs are very close to 1 in both tests, and they do not possess any systematic pattern, which provides empirical support for the predictive power of estimation model (3) for the break-even time coefficient. A summary of the results can be seen in Table 3. The results are classified according to different areas, elongations and the number of nodes. Entries in the table are the averages of the ratios between NT's of conjugate runs.

Table 3: Test results of the estimation function for break-even time coefficient c. The numbers on the table are the ratios between the NT's of conjugate runs.

A	n	$v/h=1$	$v/h=4$	A	n	$v/h=1$	$v/h=4$
1	10K	0.9980	1.0076	64	10K	0.9981	1.0067
	12K	1.0034	1.0088		12K	1.0038	1.0066
	14K	1.0052	1.0106		14K	1.0058	1.0000
	16K	1.0031	0.9998		16K	1.0046	0.9988
	18K	1.0053	1.0000		18K	1.0063	0.9984
4	10K	0.9949	1.0042	324	10K	0.9952	1.0073
	12K	1.0047	1.0074		12K	1.0056	1.0061
	14K	1.0040	0.9998		14K	1.0042	0.9973
	16K	1.0048	1.0006		16K	1.0019	0.9989
	18K	1.0073	0.9986		18K	1.0057	0.9999
16	10K	0.9965	1.0059	1024	10K	0.9960	1.0068
	12K	1.0040	1.0070		12K	1.0036	1.0052
	14K	1.0042	0.9987		14K	1.0040	0.9997
	16K	1.0029	0.9985		16K	1.0021	0.9988
	18K	1.0061	0.9995		18K	1.0054	1.0021

To use the CIP approach in a different scenario with a preferred α, we can use (3) to estimate the smallest applicable time coefficient, i.e., the fastest computation or slowest travel speed for which the CIP approach can still decrease the completion time. If the real time coefficient $c' \in \left[\frac{T^{2opt}_{G_1}+T^{2opt}_{G_2}-T^{2opt}}{C^{\mathrm{NN}}+C^{2opt}-C^{\mathrm{NN}}_{G_1}}, \frac{T^{2opt}_{G_1}}{C^{\mathrm{NN}}_{G_2}+C^{2opt}_{G_2}}\right]$, then CIP decreases the total completion time by $\left(\frac{c'C^{\mathrm{NN}}+c'C^{2opt}+T^{2opt}}{c'C^{\mathrm{NN}}_{G_1}+T_{G_1}+T_{G_2}}-1\right)100\%$.

2.8 Summary

In this chapter, we introduced a computation-implementation parallelization (CIP) approach to solving problems where the goal is to minimize the time from getting the problem instance to completing the implementation of its solution. By embedding computation into the implementation, we can decrease the total completion time. Focusing on a min-completion-time variant of TSP that we name TSP Race, we proposed and evaluated four methods of partitioning the graph for CIP, and verified

that CIP is a beneficial approach in many instances. We then described a model for predicting a priori which instances would benefit from using CIP, and showed through computational testing that it has high accuracy.

The ideas presented in this chapter are more generally applicable; in the next chapter, we use this approach to solve VRP instances that arise in parcel delivery.

2.9 Appendix: A Geometric Analysis of 2*opt*

Our experimental results on rectangular-shaped graphs where nodes are uniformly randomly distributed show that behavior differs with the graph elongation and the number of nodes. However, in all of them we observe that on the final tour at least 66% of the edges were present on the initial tour as well. The plots in Figure 14 show the fraction of identical edges in the initial and final tour for different numbers of nodes in unit-area rectangular graphs with different elongation factors.

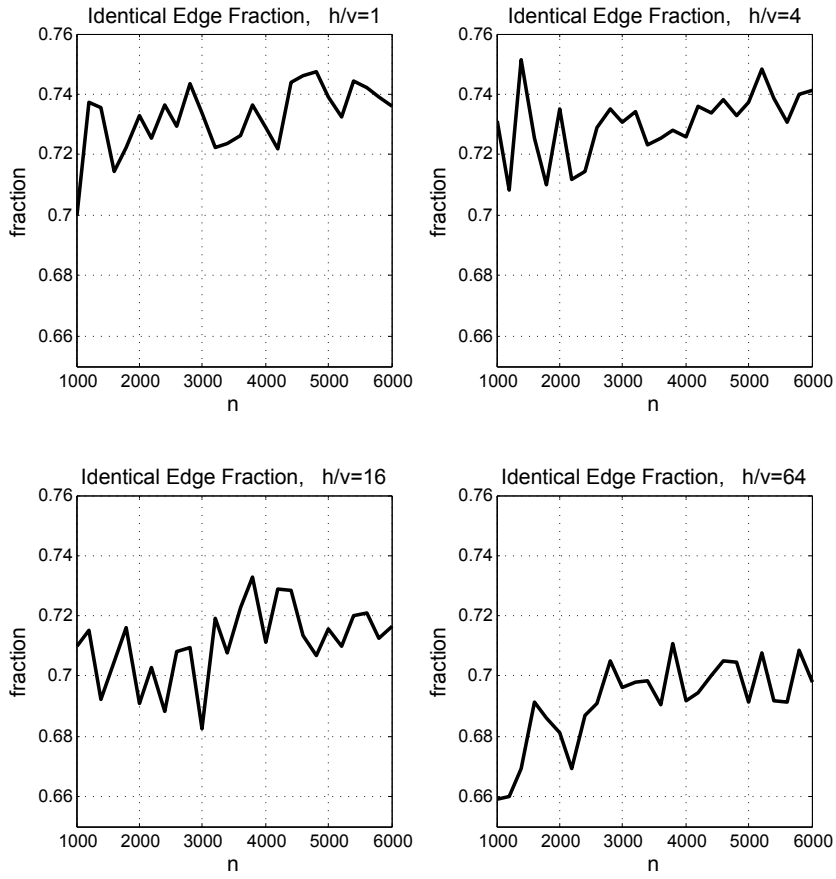


Figure 14: Fraction of identical edges in initial NN tour and final 2*opt* tour.

The large number of edges that are identical would seem to favor the use of $\mathrm{CIP}_{\mathrm{NN}}^{2opt}$. However, what is more important is the number of consecutive edges that are not swapped. In Figure 15, assume that we start to travel on e_1 and we fix only e_1 and e_2 as our first path. If 2*opt* keeps either e_1 and e_2, or e_1 and e_3, on the original tour, although the number of identical edges is the same in both, the first scenario is better. In the second scenario, edge e_2 would be already fixed preventing the improvements by 2*opt*.

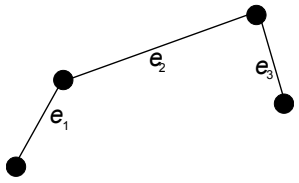


Figure 15: Fixing e_1 and e_2 on the first path.

In the experiments described above, on average of 2.9% of the consecutive edges are identical in the initial and the final tour obtained after 2*opt* improvements. So, if we could start traveling from the first node of these consecutive edges and complete the 2*opt* calculations in the time it takes to travel on 2.9% of the total edges, we could get the benefit of 2*opt* with the computation time of NN.

*2.9.0.1 NN/**2opt** Identicality Region*

Motivated by the number of consecutive unswapped edges in the initial tour, in this section we study the configurations that prevent edges from being swapped by the 2*opt* algorithm. This allows us look at the initial tour and estimate the likelihood of the first edges being identical in the 2*opt* solution. For this purpose, we make a regional analysis of the identicality condition, derive a parametric expression for this region, and do sensitivity analysis. Englert, Rögling, and Vöcking [30] made a probabilistic analysis of a single swap. Chandra, Karloff, and Tovey [15] developed results on the solution quality and the expected number of iterations for 2*opt*, looking at the whole graph.

In Figure 16, node j is the successor of node i in a tour, and a is the length of the shortest edge incident to node i in the graph.

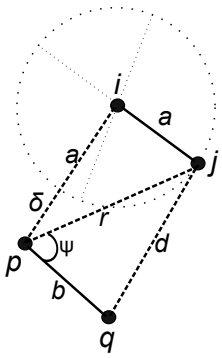


Figure 16: Consider swapping edges (i, j) and (p, q) when i is the starting node, j is the closest node to i and q is the successor of node p on the current tour.

Now consider that there is another node p which is connected to node q with an edge of length b on the current tour, and $2opt$ checks whether (i, j) should be swapped with (p, q). We are looking for the conditions that prevent the two edges from being swapped. The required condition for this is $a + b \leq (a + \delta) + d$, or $b \leq \delta + d$. Since we assume Euclidean metric TSP, we can replace d with $\sqrt{b^2 + r^2 - 2rb\cos\psi}$. When the configuration of these four nodes on the tour satisfies the following inequality, there is no swapping between (i, j) and (p, q):

$$b \leq \delta + \sqrt{b^2 + r^2 - 2rb\cos\psi} \tag{4}$$

We can analyze this under two cases, to find sets S_1 and S_2 of (b, ψ) pairs for which swapping will not occur.

Case 1: $b \leq \delta$

In this case, inequality (4) is satisfied at all possible angles ψ. The set $S_1 = \{(b, \psi)) : b \leq \delta, 0 \leq \psi \leq \Pi\}$ contains all such (b, ψ) pairs.

Case 2: $b > \delta$

$$b \leq \delta + \sqrt{b^2 + r^2 - 2rb\cos\psi} \tag{5}$$

$$b - \delta \leq \sqrt{b^2 + r^2 - 2rb\cos\psi} \tag{6}$$

Since both sides of (6) are positive, taking the square of both sides and rearranging terms yields

$$\cos\psi \leq \frac{r^2 + 2b\delta - \delta^2}{2rb}. \tag{7}$$

So, $S_2 = \{(b, \psi)) : b > \delta, \cos\psi \leq \frac{r^2+2b\delta-\delta^2}{2rb}\}$.

What happens if all the parameters remain the same, but only b increases? The derivative of $\cos\psi$ satisfying (7) at equality with respect to b is

$$\frac{-r^2 + \delta^2}{2rb^2}. \tag{8}$$

By the triangle inequality for $\triangle ijp$ in Figure 16, $\delta \leq r$; hence the derivative (8) is non-positive. That means with an increasing distance between nodes p and q, the required minimum angle ψ for no swapping between (j, p) and (p, q) is nondecreasing. When node q gets farther from node p, the no-swapping region for q requires it to also be farther from node j so that $\mid ij \mid + \mid pq \mid \leq \mid ip \mid + \mid jq \mid$. This forces ψ to get bigger and node q must be farther from node j. The picture in Figure 17 shows how b and ψ determine the no-swapping region. Given the three nodes (i, j, p), if node q, which is the successor of node p on the current tour, falls in the shaded region, swapping cannot occur between (i, j) and (p, q). As it is seen in the picture, for each b denoting the distance between nodes p and q, there exists a specific angle $\angle jpq$ denoted by ψ, beyond which swapping cannot occur. The end points of the partial circles in Figure 17 are represented by unique (b, ψ) pairs. These points constitute a curve, which we name the *boundary curve*.

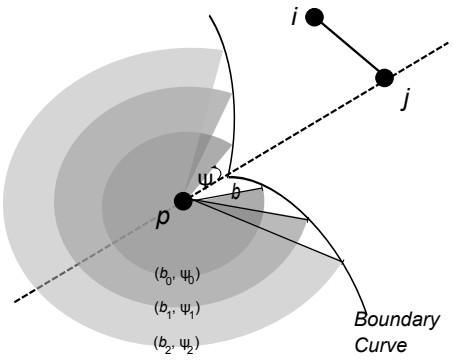


Figure 17: Swapping-Free Region.

For any node q on the boundary curve for which $\mid pq \mid = a + \delta$ and $\angle jpq = \psi$, (7) holds at equality. For a given distance b between nodes p and q we will call the angle ψ satisfying (7) at equality the critical ψ and denote it as ψ^c.

The expression of the boundary curve can be rewritten as follows:

$$\cos\psi^c = \frac{r^2 + 2b\delta - \delta^2}{2rb} = \underbrace{\frac{r^2 - \delta^2}{2r}}_{\kappa_1}\frac{1}{b} + \underbrace{\frac{\delta}{r}}_{\kappa_0} \tag{9}$$

For the sake of simplicity, we name $\frac{r^2-\delta^2}{2r}$ as κ_1, and $\frac{\delta}{r}$ as κ_0. It is easy to see that $\kappa_1 = \frac{r(1-\kappa_0^2)}{2}$. Equation (9) becomes

$$\cos\psi^c = \kappa_0 + \frac{\kappa_1}{b}. \tag{10}$$

We replace the parameter $\mid pj \mid = r$ with an angle $\angle jip = \theta$, and express r, κ_0, and κ_1 as a function of a, δ, and θ:

$$r = f_0(a, \delta, \theta) = \sqrt{a^2 + (a+\delta)^2 - 2a(a+\delta)\cos\theta} \tag{11}$$

$$\kappa_0 = f_1(a, \delta, \theta) = \frac{\delta}{r} = \frac{\delta}{f_0(a, \delta, \theta)} \tag{12}$$

$$\kappa_1 = f_2(a, \delta, \theta) = \frac{r(1-\kappa_0^2)}{2} = \frac{f_0(a, \delta, \theta)(1 - f_1(a, \delta, \theta)^2)}{2} \tag{13}$$

Throughout the analysis of the no-swapping region for node q, we have considered the effect of distance b, which has a fairly intuitive influence. Now, we will show a

sensitivity analysis for ψ^c with respect to the other three parameters: the distance between nodes i and j, the distance of node p to the ball $B(i,a)$, and the angle $\theta = \angle jip$.

2.9.0.2 Sensitivity of critical angle ψ^c

The analysis assumes that $b > \delta$, otherwise ψ^c is always 0.

1. **Sensitivity to a, the distance between nodes i and j:** The partial derivative of $\cos\psi^c$ with respect to a gives us

$$\begin{aligned}\frac{\partial \cos\psi^c}{\partial a} &= \frac{\overbrace{(1-\cos\theta)(2a+\delta)}^{\geq 0}(0.5\delta^2 - b\delta + 0.5(2a^2+2a\delta+\delta^2-(2a^2+2a\delta)\cos\theta))}{\underbrace{b(2a^2+2a\delta+\delta^2-(2a^2+2a\delta)\cos\theta)^{1.5}}_{>0}} \\ &\approx \delta(\delta-b) + \underbrace{a(a+\delta)(1-\cos\theta)}_{\geq 0}\end{aligned}$$

   The term $(1-\cos\theta)(2a+\delta)$ in the numerator is always nonnegative. The denominator, which can be rewritten as $b(2a(a+\delta)(1-\cos\theta)+\delta^2)^{1.5}$ is positive. Hence, the first derivative has the same sign as $\delta(\delta-b)+a(a+\delta)(1-\cos\theta)$, if it is not equal to 0. The derivative is nonnegative only when the distance between nodes p and q is bounded above by

$$\frac{a(a+\delta)(1-\cos\theta)+\delta^2}{\delta}. \tag{14}$$

   There are two cases we can consider. The first is when the distance between nodes p and q is less than (14). This corresponds to region (II) in Figure 18. In this region, nodes p and q do not fall on opposite sides of ball $B(i,a)$, since the edge length connecting them is not long enough to cross the ball. Because of that, ψ^c can get small, resulting in a wider no-swapping region. The second case, where $b < \frac{a(a+\delta)(1-\cos\theta)+\delta^2}{\delta}$, corresponds to region (I) in Figure 18. In

that region, when j is close to i, swapping is already difficult, and so ψ^c can get small. As node j gets farther from node i, the no-swapping region requires node q to be far enough from node j as well. This is ensured through an increase in ψ^c. In the first plot in Figure 18, ψ^c increases until $a = 0.536$, which is the stationary point for $\cos\psi^c$, then it starts to decrease.

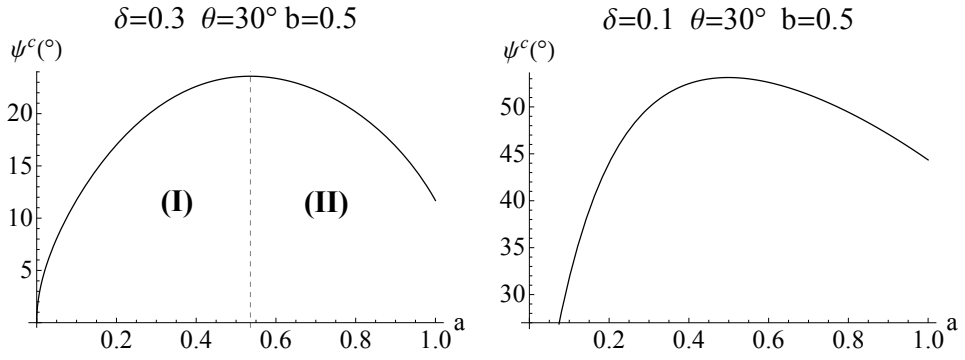


Figure 18: Change in ψ^c with respect to the distance between nodes i and j (a).

2. **Sensitivity to δ, the distance between p and $B(i,a)$:** Recall that δ is the difference between $\mid ij \mid$ and $\mid ip \mid$. In contrast to the nonintuitive relation between a and ψ^c, δ has a predictable effect on ψ^c.

   The partial derivative of $\cos\psi^c$ with respect to δ is a positive multiplier of $[(a^2+ab)-(a^2+a\delta)\cos\theta+ab+\delta b]$ which is always positive, because $b > \delta$. This implies that ψ^c is decreasing for increasing values of δ. This is expected, since bigger values of δ places node p farther from $B(i,a)$ providing a wider region for node q.

   Two example plots for the relation between ψ^c and δ are depicted in Figure 19. The critical angle ψ^c for no swapping decreases in both; however the acceleration is different due to the difference in other parameters.

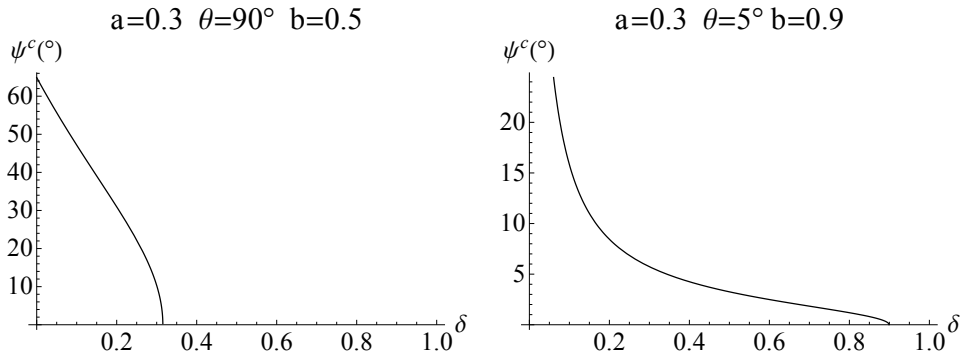


Figure 19: Change in ψ^c with respect to the distance between node p and $B(i, a)$, (δ).

3. **Sensitivity to θ, the angle $\angle jip$:** The derivative of $\cos\psi^c$ with respect to θ is positive only when the following inequalities hold:

$$\cos\theta < \frac{4a + a^2 - \delta^2}{a^2 + \delta + 2a\delta} \tag{15}$$

$$b < \frac{a^3(1-\cos\theta) + 2a^2(2 - \delta\cos\theta) - a\delta\cos\theta}{a\delta + \delta^2} \tag{16}$$

The no-swapping region becomes larger as node p is placed farther from j due to a larger angle θ, because node q is connected to p with a short arc whose length is bounded above as in (16). This flexibility is reflected by a decrease in ψ^c which can be seen in the first plot of Figure 20. In the second plot, we keep all the parameters the same but increase the distance between p and q from 0.2 to 0.9. We observe that the minimum angle ψ^c defining the partial circle for no swapping cannot be zero even when nodes j and p are exactly on the opposite sides of i, i.e., $\angle jip$=180°. This is because the large distance between p and q makes this edge undesirable for the tour. If they are to be placed in the no-swapping region, ψ^c should be wide enough to ensure that node q is also further from node j to prevent any incentive to include $\mid jq \mid$ in the tour.

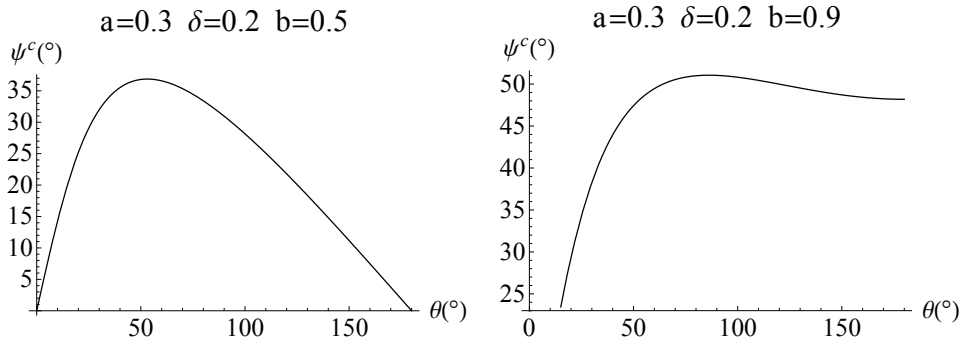


Figure 20: Change in ψ^c with respect to the angle $\theta = \angle jip$.

Chapter III

A COMPUTATION - IMPLEMENTATION PARALLELIZATION APPROACH TO THE COMPUTATION-TIME LIMITED VEHICLE LOADING AND ROUTING PROBLEM

3.1 Introduction

Growth in the size of logistics operations has led to correspondingly larger and more challenging instances of routing problems. At the same time, the integration of routing decisions with other operational issues has become an area of interest. Although these integrated problems provide solutions that are better for overall operations than optimizing each decision separately, they are computationally much more demanding, especially for large instances. In practice, it may be difficult to allocate enough computation time to solve integrated problems; we may not be able to reach optimal or near-optimal solutions on large instances even for the basic (non-integrated) version of the problem.

In this chapter, we address such an integrated problem that arises in large delivery operations in which small, time-sensitive orders are placed at a warehouse and delivered by truck to customers. The base routing problem is a Capacitated Vehicle Routing (CVRP), with the additional complicating operational issue of the time required to load the orders onto the trucks. In practice, the two tasks are done one after the other.

At a predetermined cutoff time (the latest time an order can be placed to be on the next set of delivery routes), vehicle routes are calculated for all orders placed

since the previous cutoff. Then, the trucks are each loaded with their deliveries. At facilities that use fleets of trucks to make deliveries to a large number of customers with a high throughput rate, both route computation and vehicle loading can take significant amounts of time. Customers want cutoff times as late as possible. Late order cutoff time increases the customer satisfaction, since more customers can be promised that their orders will be placed on the first-leaving trucks. However, late cutoffs also limit the available time to compute good routes.

When the operating schedule in a warehouse is too tight to allow enough computation time for good routing, we propose a different approach for the routing problem. In this new setting, in addition to the usual constraints of the routing problem we address an implicit constraint on the computation and loading time. We call this new setting the Computation Time-Limited Vehicle Loading and Routing Problem (CTL-CVRP). To solve this problem, we implement a Computation-Implementation Parallelization (CIP) approach, and parallelize the computation with loading to create additional time for computation.

By implementing our CIP approach, we can embed almost all the computation in the loading; in some cases it is possible to decrease the computation-only time to only a few minutes. This can benefit us in different ways. First, we might improve the routes without changing the order cutoff or truck dispatching times. Or, we can tradeoff some of the improvement in the routing and use it either to delay the cutoff time or to move the truck dispatching to an earlier time.

The rest of this chapter is organized as follows. In Section 3.2, we present a brief review of VRP, its variants and integration with other problems. In Section 3.3, we discuss the methods to embed computation time into loading. In Section 3.4, we present and discuss computational results. In Section 3.5, we propose a modification on Toth and Vigo's [67] tabu search algorithm. In Section 3.6, we present some final remarks.

3.2 Literature Review

In recent years, interest in integrated routing problems has increased. In this context, VRP has been integrated with lot-sizing (e.g., [17], [24], [10], [35]), scheduling (e.g., [16], [68]), 2-dimensional packing (e.g., [46], [70], [37], [32]), 3-dimensional packing (e.g., [36], [33], [9]), cross-docking (e.g., [53]), etc. The structure of these integrated problems generally combines the two decisions in one model to find a globally better solution rather than focusing on one at a time. To the best of our knowledge none of the models in the literature has integrated VRP with the constraint on the time spent in computation and loading, other than simply truncating a heuristic at the end of the allowed computation time. For the simpler TSP, in Chapter 2 introduce a CIP approach to integration of computation and implementation issues and show its potential utility; in this chapter, we use a similar CIP approach.

Our approach requires a base method for solving large VRP's. To solve VRP's of a size relevant for CTL-VLRP, exact VRP methods are too slow in practice. For surveys on exact methods, we refer to Baldacci et al. [4], Barnhart et al. [5], Laporte [52] and Toth and Vigo [66]. Our goal is to provide practical solutions; therefore we focus on heuristic methods. There are many VRP heuristics beginning with that of Clarke and Wright [21]. Toth and Vigo [66], Laporte [52], Barnhart et al. [5] provide comprehensive surveys of heuristics. Bräysy et al. [11] focus on more recent basic traditional route construction and local search algorithms; Bräysy et al. [12] focus on metaheuristics.

To implement CIP, we considered recent heuristic methods which are shown to produce good results. One of these heuristics is the granular tabu search (GTS) developed by Toth and Vigo [67]. This algorithm is very competitive and fairly straightforward to implement. To reduce the computation time, given the original graph, they create a new sparse graph eliminating long arcs which are unlikely to be

included in a good solution. Another heuristic we considered is the evolutionary algorithm developed by Mester and Bräysy [55]. Computational results show that their two-phase evolutionary algorithm is very competitive compared to others in the literature. Kytojoki et al. [51] also developed a heuristic for large scale problems. Nagata [57] extended an evolutionary algorithm which was originally developed for TSP to CVRP by penalizing the moves causing infeasibilities. Since the most time consuming part of the algorithm is the local search, he later applied 5 different local search limiting strategies (Nagata and Bräysy [58]) to improve computational performance while maintaining the solution quality. Among these heuristic methods, we select Toth and Vigo's [67], because it is simple, flexible, and provides very competitive results in short amount of time.

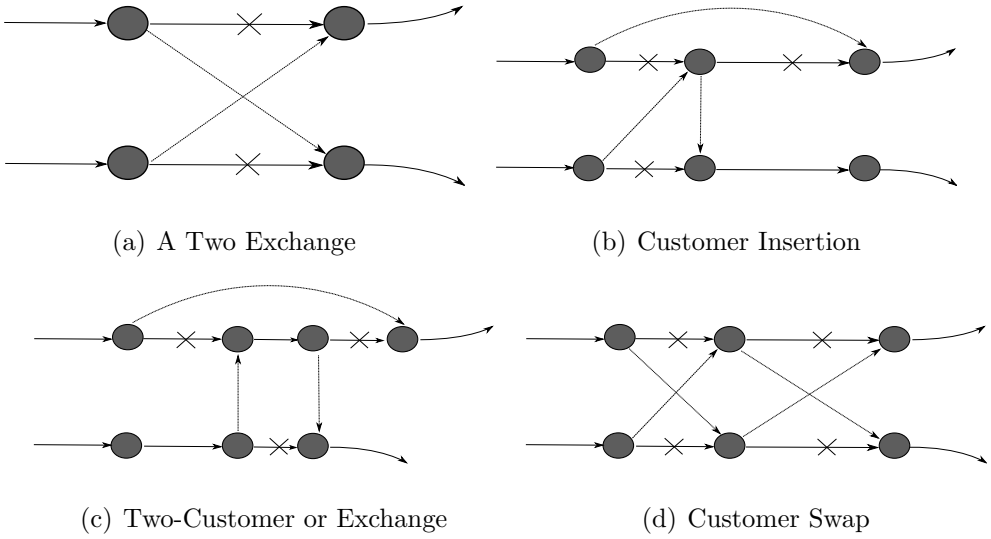
3.3 Solving CTL-CVLRP

In this section, we describe our approach to solving CTL-CVLRP. We begin with a problem instance. In this problem setting, information is complete. A graph $G = (V, A)$ is given, where $V = \{0, 1, \ldots, n\}$ is the set of vertices and A is the set of arcs available for travel. Vertex 0 corresponds to the single depot. Vertex set $V \setminus \{0\}$ is the set of n customers. c_{ij} denotes the distance between vertices i and j; distances are Euclidean. There are K trucks each with maximum capacity C. Each customer i's delivery requires d_i space on a truck. Loading a delivery of size s requires δ_s time. Computation and loading can begin at the cutoff time $t = 0$, and must be completed by time $t = T$.

The objective is finding the minimum-cost K routes from the depot such that each customer is visited exactly once, and the capacity constraint is obeyed.

Toth and Vigo's [67] GTS algorithm starts with an initial solution constructed by Clarke and Wright's [21] savings heuristic. Tabu search is then run using four types of neighborhood searches. These are two exchange, customer-insertion, two-customer

and customer swap. Figure 21 illustrates each move by one example.



(a) A Two Exchange (b) Customer Insertion

(c) Two-Customer or Exchange (d) Customer Swap

Figure 21: Four neighborhoods used by GTS.

Toth and Vigo [67] observed that the average arc length in a complete graph is much more than the average arc length in the optimal solution. Therefore, long arcs are unlikely to be present in a good solution. Based on this observation, given a graph G, they create a new, sparser graph by omitting the long arcs. They define a long arc as the one whose length is larger than some threshold ν, where

$$\nu = \beta \frac{z^{'}}{(n+K)}. \tag{17}$$

$z^{'}$ is the tour length of a solution found by a heuristic, and β is a sparsification factor that is selected computationally. Tabu search is performed on the sparse graph. The details of creating the sparse graph are explained in [67].

Although tabu search algorithms are successful heuristic methods for many routing problems, the required computation times can get quite large for large scale instances. Toth and Vigo [67] recorded the computation time to find the best solution for the instance E421-41k as 43.01 minutes. This instance has just 420 customers and 41 trucks. Our focus is on large scale instances with 1,000 or more customers. Such

instances are computationally demanding. Due to the time limitations, it is not always possible to allow enough computation time for commonly-used algorithms to find the best possible solutions. Therefore, solving these problems using CIP in the CTL-CVLRP context rather than just CVRP can be a more suitable approach.

3.3.1 A CIP Approach

The way CTL-CVLRP is usually addressed in practice is in two phases. First, once the cutoff time has passed, a delivery (routing) solution is computed. Then, the packages are loaded onto trucks according to the solution, and the trucks are dispatched. In our CIP approach, we first make a short computation, and fix some customers onto trucks. While loading the items belonging to those customers, we make additional computations for the rest. During the additional computations, we may again (several times) fix and load some deliveries while continuing to compute a solution for the rest. Once an item is loaded onto a truck, we do not relocate it. Other decisions made during the computation are not permanent until they are implemented (loaded). Figure 22 shows how the CIP approach works. With the exception of computing the first deliveries to be fixed, and loading the final set of deliveries to be fixed, each computation phase is done in parallel with a loading phase, and vice versa. In some cases the computation might finish before its parallel loading operations (e.g., between the first and second fixings in the figure), or the opposite might be true (e.g., between the second and third fixings).

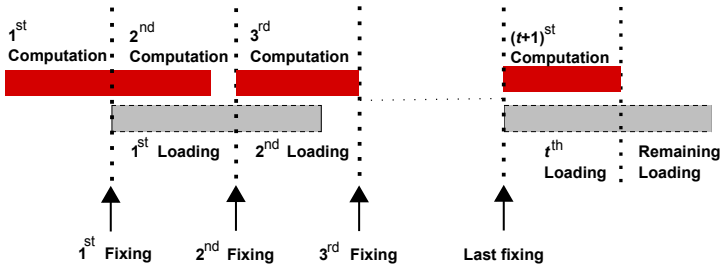


Figure 22: Computation-Loading Parallelization.

The potential benefit of CIP is that, although, some deliveries are fixed early, more computation time is available to better optimize the remaining deliveries. This can result in better delivery routes overall; alternatively the order cutoff time can be extended or trucks can be dispatched earlier with solutions of similar quality. On the other hand, fixing some deliveries early could result in worse overall solutions. This issue is discussed later in Section 3.4.

The sequence of a customer on its route can be important for loading purposes. If the items are heavy or large, then Last-In-First-Out (LIFO) loading is likely to be used. Otherwise, it would require too much effort to retrieve the item at a customer location. If the items are small boxes, then we can assume random-access trucks. Our fixing policies will differ depending on whether trucks are LIFO or random access.

3.3.2 Customer Fixing Policies

LIFO and random access loading will require different fixing policies. How many customers we fix is determined by a parameter α, as it will be explained in detail below. In general, small values of α means fixing a small number of customers, though the mechanism will differ from policy to policy. In Section 3.4, we compare our policies in order to suggest the best.

We classify our customer fixing policies in two groups. If we make the customer-fixing decision depending only on the current status of the routes, we call these policies *simple fixing policies*. Alternatively, some of our fixing rules use the information from tabu search history. They are called as *information-collecting fixing policies*.

3.3.2.1 Policies for LIFO Loading

When the LIFO loading policy is used, we can only fix the last customers on a route, since the items placed in the front section of a truck will be delivered last. Otherwise, it would require fixing some items in the middle or at the rear of the truck, which is not meaningful.

1. **Simple fixing policies**

   Simple fixing policies only consider the position of customers on the routes. There are potential benefits of fixing customers on each truck, and of concentrating the fixing on fewer trucks. Therefore, we test both types of policies.

   [P1] Multiple-truck simple fixing: This policy tries to take advantage of the fact that if the last customers of routes are close to the depot, they may not be swapped by GTS. Under this policy, starting from the last customer on route i, we fix αn_i consecutive customers, where $i = 1, \ldots, K$ and n_i denotes the number of customers on truck i. Then, during the tabu search any moves trying to change the position of these fixed customers are not allowed.

   [P2] Minimum-load truck fixing: Until αn customers are fixed, we iteratively choose the truck that has the smallest load and at least one unfixed customer, and we start fixing from the last unfixed customer. We attempt to fix αn of the customers, but it there are fewer than αn unfixed customers on the truck, we fix customers on the truck which has the second minimum-load, and repeat until αn customers are fixed.

   [P3] Maximum-load truck fixing: This policy is similar to [P2], but this time we start fixing the customers from the maximum-load truck. The selected trucks must satisfy the capacity and the travel distance constraints; otherwise, we skip this truck.

2. **Information-collecting fixing policies**

   Simple fixing policies are based on the expectation that the last customers on the current routes will be mostly kept as they are during the tabu search. An alternative to making the fixing decisions depending on the current state of the

routes is using the tabu search history to determine how to do the fixing. During the search, to prevent the tabu moves, we store the last iteration number when an arc was involved in a move. This number determines the *age* of an arc. If it is small, the arc is old. Our information-collecting policies use the age of the arcs on the current routes to fix customers.

We use the tabu search history in four different ways.

[P4] Multiple-truck oldest-arc fixing: With this policy we fix the last customers of trucks which have not been swapped for a long time during the search. To implement this policy, we first create an array of size K, which is the number of trucks. The initial elements of the array are the last unfixed customer of each route. To decide which customer to fix, we search through the list and fix the customer which corresponds to the oldest arc among the K of them. Then we replace this customer in the array with its predecessor customer on the route. We repeat this until we fix αn customers.

[P5] Multiple-truck youngest-arc fixing: The procedure is the same as [P4], but this fixing rule selects the the arcs which are involved in an improvement move most recently, i.e., youngest arcs.

[P6] Oldest-truck fixing: During the search, if the customers of a route (truck) have not been involved in swapping moves recently, it may be more likely that this route does not include any significant improvement moves. To be able to exploit such situations, we compute the average age of the arcs of each truck at the time of the fixing, and we start fixing customers from oldest truck to the newest until we fix αn customers.

[P7] Youngest-truck fixing: This fixing policy is the same as [P6], but this time we fix customers starting from the truck whose average age of

arcs is the youngest.

3.3.2.2 Policies for Random Access Truck Loading

When the items are small and can all be accessed in the truck at any time, we do not have to follow the LIFO loading rule. Therefore, once we know which truck will deliver the order of a specific customer, we can load the item onto that truck without needing information about the sequence of the customers on the route.

An alternative CIP approach on random-access routes would be to determine which customers will be visited by which truck without optimizing the routes, i.e., separating customers into zones. Then, load their items onto trucks, and perform the route optimization while traveling as we have done in Chapter 2. However, this imposes a strong constraint on the routes; once a truck is loaded with the items and leaves the depot, we can only perform improvement moves within its own customers. Therefore, we lose the opportunity of switching customers between the trucks.

1. **Simple fixing policies**

   [**P8**] In random-access loading, the simple fixing policy only considers the closeness of a customer to its predecessor on the current route. At the time of fixing, we base our decision on the distance between consecutive customers as follows. If this distance is less than $\alpha\nu$, where ν is the threshold value in equation (17), we fix these two customers on the current truck, and no longer allow GTS moves that would separate those customers.

   Note that the use of customer fixing parameter α for [P8] is different than the others. Here, α is the parameter we use to determine which customers are considered as close to each other. As α decreases, we fix fewer customers. However, in other policies α directly determines the number of customers to fix.

2. **Information-collecting fixing policies**

   Information-collecting fixing policies for random-access loading trucks work in a similar way to the ones corresponding to LIFO loading. However, since the sequence of the customers on a truck does not have to follow the sequence of the customers on routes, we are more flexible. For random access loading, we tested two different information-collecting fixing policies.

   [P9] Oldest-arc fixing We sort the arcs of the current solution with respect to their age during the tabu search. Then, starting from the oldest arc we fix the starting and ending nodes until αn customers are fixed.

   [P10] Youngest-arc fixing This is same as [P9], except that we fix customers from youngest to oldest.

3.4 Comparison of CIP Policies and Pure GTS

To compare CIP and pure GTS, we coded the algorithm in C and we ran computational experiments on randomly generated instances. In our instances, we assume that customers are uniformly randomly distributed on a square, and the single depot is located at the center. Demands of customers are equal, and the total truck capacity has 10% slack.

Before going into the details of the test settings, we first introduce some notation:

Notation:

- n: number of nodes (customer locations),
- K: number of trucks,
- t_l: how long (in seconds) it takes for single customer's order to be picked and loaded onto a truck (assumed to be the same for all customers) divided by the number of available pickers/loaders that can work in parallel,

- F: percentage of customers we fix in total,
- m: number of computation (and also loading) steps,
- α: $F/(m-1)$, percentage of customers we fix in each fixing step,
- T_0: computation time allowed in standard (non-CIP) operation,
- L: total loading time requires, equal to nt_l,
- T: total time spent on computation (including that parallelized with loading),
- $C[i]$: computation time allocated for step i where $i = 1, \ldots, m$.

We observed that for a given total fixing rate F, it is generally better to do the fixing in more than one step rather than fixing Fn customers at once. The reason is that, when we do multi-step fixing, the tabu search is allowed to consider more options for a longer time, because once we fix a customer, any improvement involving that customer is not permitted. Therefore, in multi-step fixing, the later steps can make more informed decisions.

When we fix Fn of the customers, the loading time of the fixed customers provides Fnt_l seconds for computation which is embedded into loading. Therefore, the total computation time we can use is $T_0 + Fnt_l$. Note that T_0 can be 0; in that case all the computation time is gained through customer fixing.

In multi-step fixing, at each each step we fix αn customers, where $\alpha = F/(m-1)$. However, when we allocate the total computation time, i.e., $T_0 + Fnt_l$, we follow 3 different time allocation rules as explained below.

1. *Equal allocation*: Total computation time is allocated equally to each of the m computation steps: $C[i] = (T_0 + Fnt_l)/m$.
2. *Linear allocation*: The total time is allocated linearly proportional to the number of unfixed customers at each computation step: $C[i] = (T_0 + Fnt_l)\frac{(1-(i-1)\alpha)}{\sum_{j=1}^{m}(1-(j-1)\alpha)}$.

3. *Quadratic allocation*: We allocate the total time proportional to the square of the number of unfixed nodes in each computation phase: $C[i] = (T_0 + Fnt_l)\frac{(1-(i-1)\alpha)^2}{\sum_{j=1}^{m}(1-(j-1)\alpha)^2}$.

The equal allocation rule partitions the total computation time into equal phases. The linear and quadratic allocation rules divide the total computation time so that the initial computation phases, where there are fewer unfixed nodes, use longer computation times compared to the later steps, where there is already less opportunity to improve.

Since each computation phase except for the first one is done in parallel with loading, it is possible that the loading time could be larger than the allocated computation time for that phase. In such a case, we extend the computation time and reduce the sum of the earlier phases' computation time by the same amount (proportional to their original allocation), so the dispatch time does not change. The pseudo code is given in Section 3.7.

3.4.1 Results for LIFO Trucks

We compared the seven LIFO fixing policies using each of the time allocation rules, on 5 different uniformly distributed customer-instances with 1000 nodes and a depot at the center, and 20 trucks.

In the first experiment (performed on identical Xeon E5620 (2.27Ghz) machines), we use T_0=60min, and loading takes 120min. That is, there are 180min between the start of computation and when the trucks leave the warehouse. In the conventional compute-first implement-later approach, 60min would be spent for computation, which is often a long enough time to find a good solution: After running the base GTS on the five instances, after 60 min the average ratio between the final solution and the initial solution is 0.924. So, this setting is more difficult for CIP to show improvement.

In our CIP approach, we divide the loading into m equal phases. In each loading phase, we load $n\alpha$ of the customers' orders, where $\alpha = 1/m$. Except for the last loading step, all of them are done in parallel with a computation phase. Therefore, the total computation time we can use is $T_0 + n(1-\alpha)t_l$, and we allocate this total computation time using the equal, linear and quadratic time allocation rules as described above.

In Figures 23, 25 and 27, we present the computational results for LIFO policies. In Figures 24, 26 and 28, we display the computation and loading phases for $\alpha = 0.2$, with their lengths drawn to scale. When $\alpha = 0.5$, there are 2 loading and computation phases; the first computation (before any fixing) is 60min, and second computation phase (after fixing half of the nodes on their current routes according to fixing policies) is also 60min. When α decreases, the total computation time we can use increases.

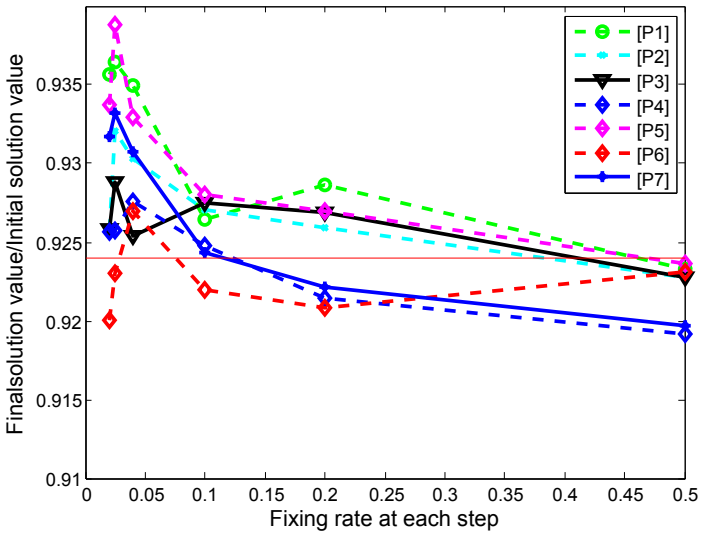


Figure 23: Customer fixing policies for LIFO using equal time allocation rule.

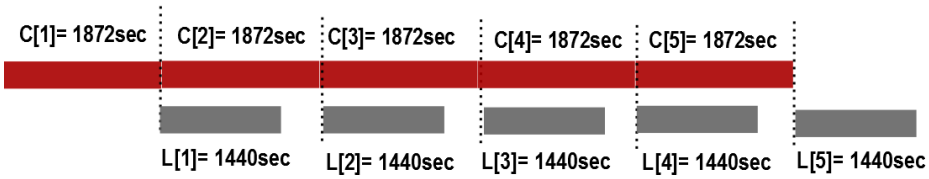


Figure 24: Equal time allocation when $\alpha = 0.2$.

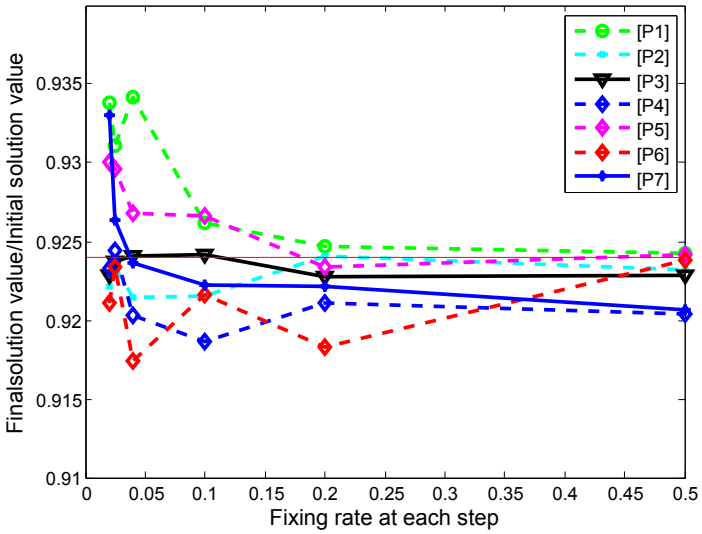


Figure 25: Customer fixing policies for LIFO using linear time allocation rule.

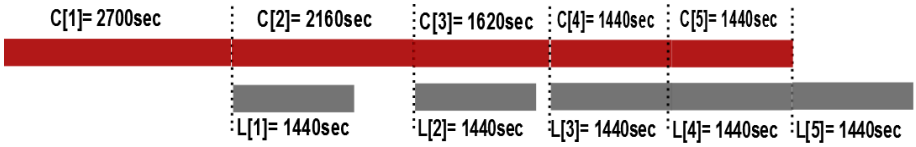


Figure 26: Linear time allocation when $\alpha = 0.2$.

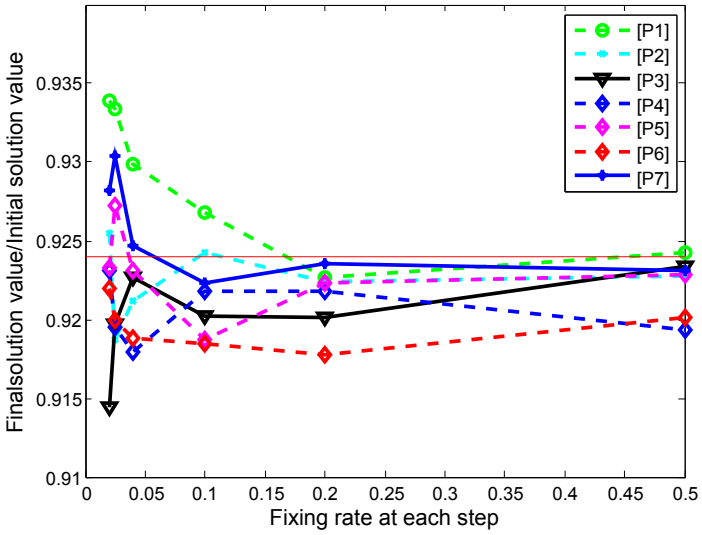


Figure 27: Customer fixing policies for LIFO using quadratic time allocation rule.

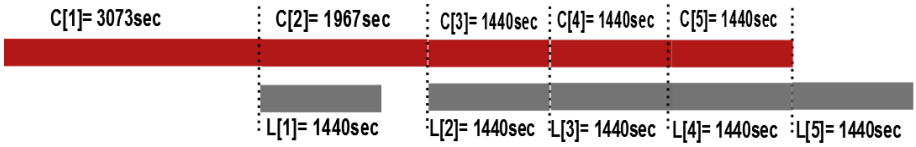


Figure 28: Quadratic time allocation when $\alpha = 0.2$.

A common observation in all time-allocation rules is that fixing customers on a single truck (or route) at each phase performs better than fixing them on multiple trucks simultaneously. The reason is that multiple-truck fixing makes the inter-route moves more difficult. [P6] seems to perform better than the others, since it uses the tabu search history while fixing the nodes and makes more informed decisions than just considering the current routes.

We also see that quadratic time allocation rule with more computation devoted to phases with more unfixed nodes, works best.

Overall, we see that although CIP is not guaranteed to give better solutions unless $\alpha = 0.5$, CIP policies are able to improve upon pure GTS. In this specific setting we chose the computation time to be quite long, and loading time to be short, which is

a tough case for CIP because GTS gives good solutions in one hour. However, we observe that by making the fixing decision in an informed way by [P6], we can still improve the solution compared to using compute-first implement-later approach.

In the second setting (performed on identical Xeon E5620 (2.4Ghz) machines), we focus on a case where the loading takes even less time, 60min. The goal of this second experiment is to test if it is possible to decrease the computation-only time allocated for the base GTS from approximately 60min to 30min while maintaining the solution quality. In this setting, we fix the 48% of the nodes in total, so the total computation time is the same at all fixing rates. As before α determines the number of nodes we are fixing in each step (we fix $n\alpha$ nodes at each step). The number of fixing steps is $0.48/\alpha$. This setting with the given parameters provides 58.8min of total computation time, with 28.8min of it created by parallelizing the computation with loading. The results of the experiments are displayed in Figures 29, 30 and 31.

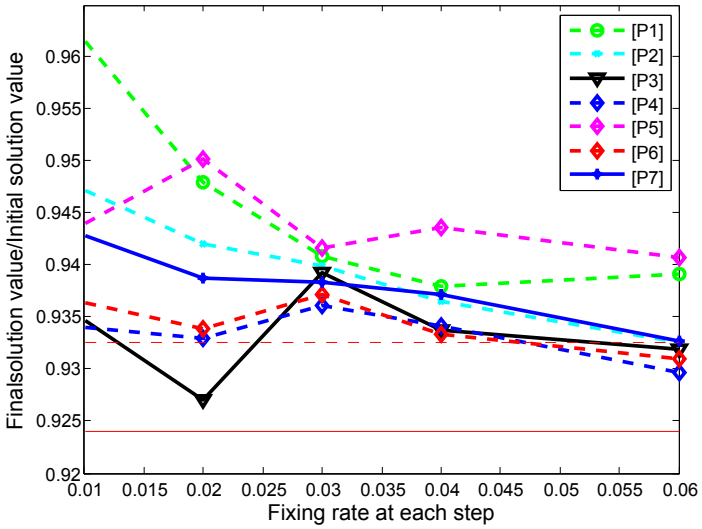


Figure 29: LIFO fixing policies when total fix rate is 48% (equal time allocation).

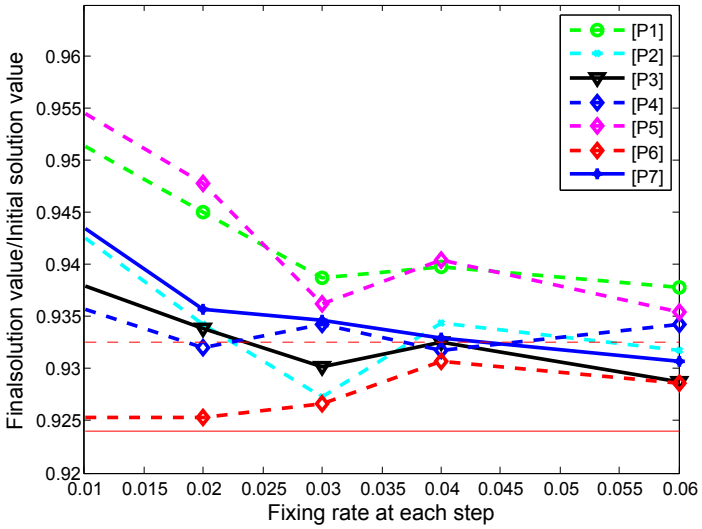


Figure 30: LIFO fixing policies when total fix rate is 48% (linear time allocation).

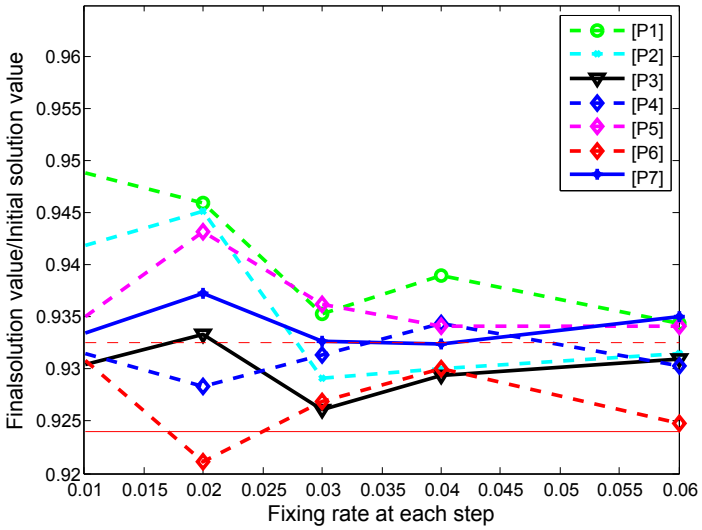


Figure 31: LIFO fixing policies when total fix rate is 48% (quadratic time allocation).

Without using CIP, the base GTS algorithm finds a solution ratio of 0.924 in 60min of computation, and 0.933 when computation is reduced to 30min. CIP allows us to create solutions closer to 0.924 while still keeping the 30min savings that can be used to improve customer service by delaying the cutoff time or releasing the trucks early.

Similar to the previous experiment, we see that the best average solution is obtained under quadratic time allocation.

3.4.2 Results for Random-Access Trucks

We also tested the random access truck CIP fixing policies using the same two settings we used for LIFO trucks. Fixing policy [P8], which fixes the nodes corresponding to the shortest arcs on the current routes, does not generate comparable-quality results with the others, and therefore we do not further consider this policy.

Figures 32, 33 and 34 display the results of the first experimental setting. In random loading, once a customer's order is fixed on a truck, it cannot be moved to another truck, but (in contrast to LIFO loading) it can be involved in intra-route improvement move. Therefore, compared to LIFO, we see a better improvement on the solution which is obtained by running the original GTS algorithm in random access case.

Our conclusion regarding the comparison of the fixing policies selecting the *old* arcs and the policies selecting the *young* arcs to fix nodes applies to random loading case as well. In all time allocation rules, we see that [P9] dominates [P10] on average.

In the random loading case, we are able to observe the differences between the time allocation rules better due to these policies being less restrictive. Since equal time allocation does not consider the number of unfixed nodes, when there are many fixings the first computation steps (in which there are many unfixed nodes) are allocated less computation time compared to the linear and quadratic time allocations. This can prevent the information-collecting policies [P9] and [P10] from gathering enough information until they start fixing. Therefore, linear and quadratic time allocations lead to better solutions. We can reach an average ratio of final solution/initial solution of 0.910 in these two allocations by generating more computation time in addition to 60min without changing the order cutoff and dispatching time, while this ratio is

0.924 if we run the original GTS for 60min.

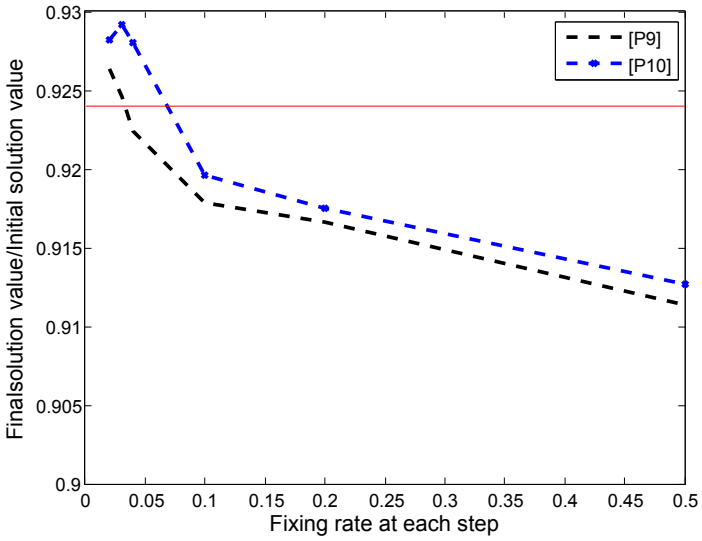


Figure 32: Customer fixing policies for random-access using equal time allocation rule.

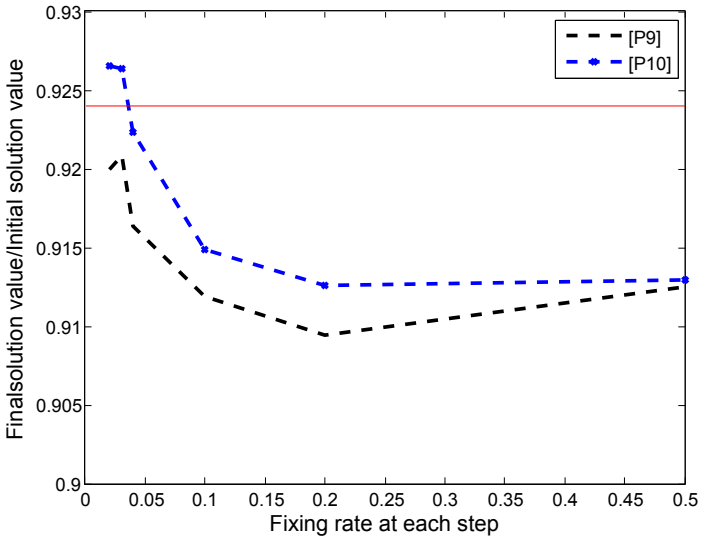


Figure 33: Customer fixing policies for random-access using linear time allocation rule.

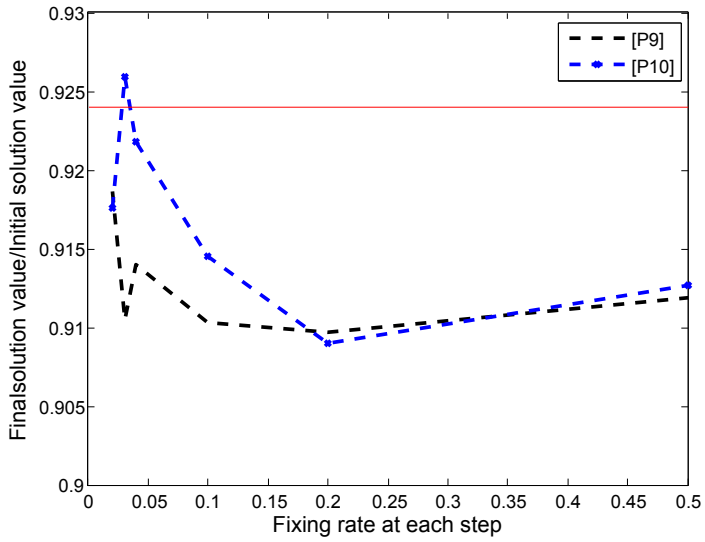


Figure 34: Customer fixing policies for random-access using quadratic time allocation rule.

Finally, we present the test results of random fixing policies on the second experimental setting. Since the total allocated computation time is less, when there are many computation steps (corresponding to small α), policies start fixing nodes very soon, which decreases the performance of [P10]. However, [P9] still maintains its higher performance and can reach the same solution quality as we could reach if we run the original GTS for 58.8 minutes.

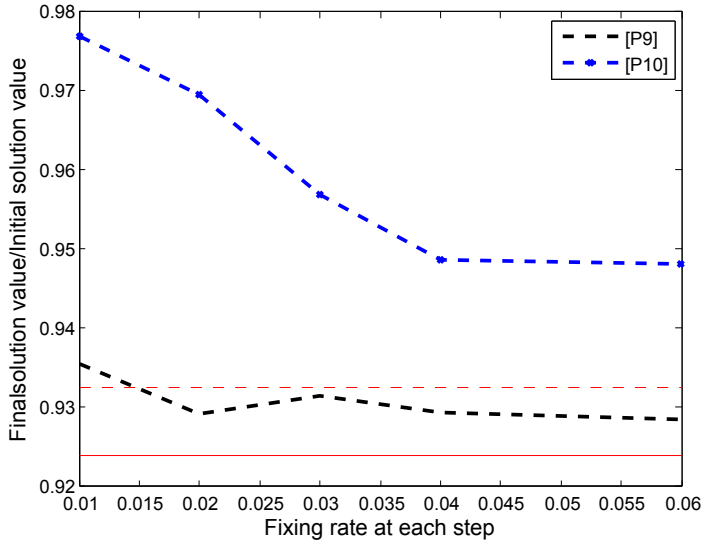


Figure 35: Random fixing policies when total fix rate is 48% (equal time allocation).

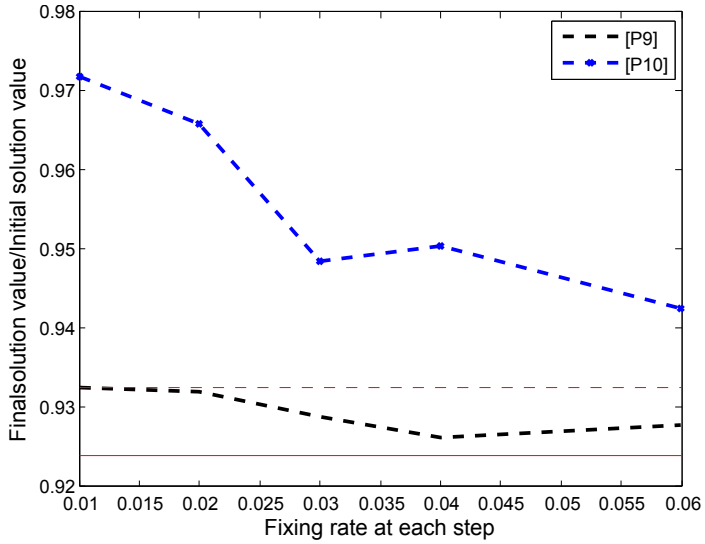


Figure 36: Random fixing policies when total fix rate is 48% (linear time allocation).

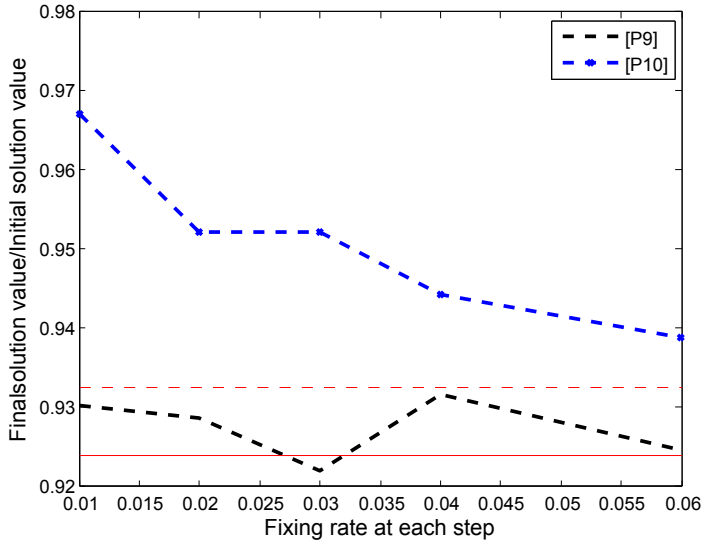


Figure 37: Random fixing policies when total fix rate is 48% (quadratic time allocation).

3.4.3 Summary

In Sections 3.4.1 and 3.4.2, we compared the CIP policies for LIFO and random loading with pure GTS using two different test settings. In the first setting, we showed that even if the present allocated computation time is long enough to get a good solution, we can still improve the solution quality by parallelizing the computation with loading without changing the order cutoff and truck dispatching times. CIP benefits more in random loading, since the fixing of a node on a truck does not prevent intra-route moves for that node in random loading. In the second setting, we showed that it is possible to set some of the computation-only time free, since by performing computation in parallel with loading we can reach the same solution quality. The time that is set free can be used to relax either to the order cutoff or the truck dispatching time.

3.5 Modified Granular Tabu Search

Toth and Vigo's [67] granular tabu search algorithm is very effective and fast, because it makes the search on a neighborhood that includes only promising moves. Its sparse graph includes the arcs that are incident to the depot, those belonging to the best solutions, and also the short ones. As mentioned before, short arcs are defined to be the ones whose length is smaller than the threshold ν. According to this definition, we draw a ball around each node with radius ν and choose all the arcs falling in this ball, as in Figure 38. We propose a different method to create the sparse graph for uniformly-distributed node locations.

In the modified GTS, we change the definition of short arcs. Instead of selecting the ones whose length is smaller than the threshold ν, we choose the ones connecting nodes i and j if $\angle i0j \leq \frac{\pi}{K}$, where 0 is the depot node and K is the number of trucks. In Figure 39, although nodes i and j are close to each other according to the original sparse graph definition, they are not considered close in the modified version. On the other hand, the arc incident to nodes i and p is not considered as a short arc in the original GTS, but in the modified GTS we also accept this arc as a short arc to the sparse graph. The mechanism of this modification is creating beams originating from the depot and forcing the trucks to travel along those beams.

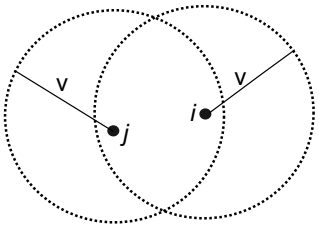


Figure 38: Sparse graph construction with respect to distance in the original GTS.

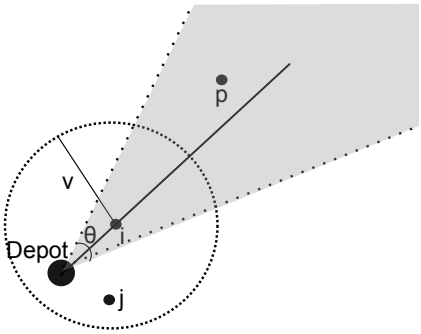


Figure 39: Sparse graph construction with respect to the polar angle in the modified GTS.

To observe the effect of this modification, we run the GTS algorithm using both the original modification method which uses the distances between the nodes and the new method which uses the polar angles between the nodes. We compared the two methods on uniformly distributed nodes.

We generated the instances as before: nodes are uniformly distributed on a square region. Customer demands are constant, and total available truck capacity has 10% slack above the total customer demand. The number of customers varies between 1,000 and 3,000 in increments of 100. The number of trucks is chosen such that the average number of customers per truck changes between 50 and 140. We allowed 60min of computation for both versions. When we compare the value of the final solution we obtain with two methods, we observe that our modification improves the performance of GTS algorithm. The modified GTS produces better results 87% of the time, and the average improvement in the final solution value compared to the original algorithm is 2%. The best improvement over the original GTS is 7% while the worst case we observed is 3% decrease in the quality of the solution. An analysis of the comparison between these two sparse-graph constructing method can be seen in the histogram plot in Figure 40. Histogram plot compiles the results of 630 different instances with the aforementioned properties.

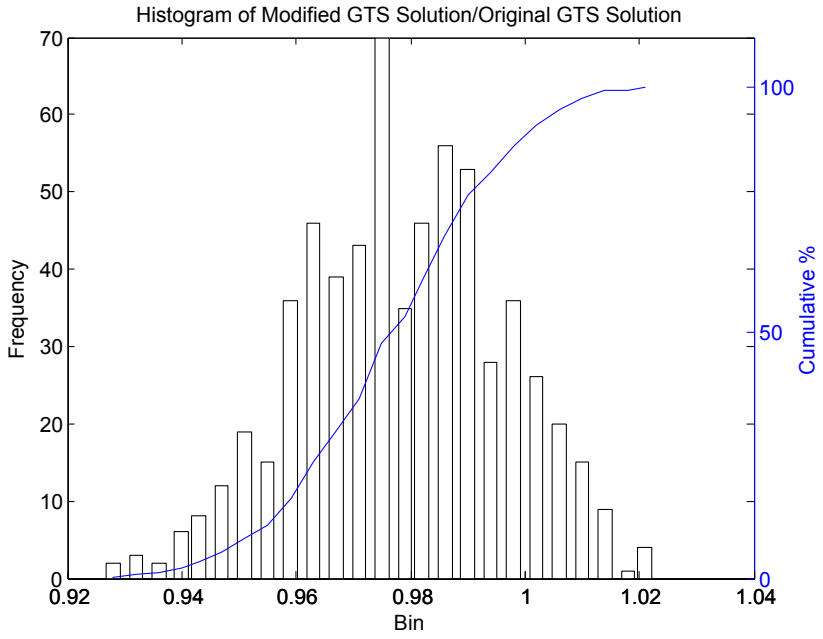


Figure 40: Histogram plot of the modified GTS solution/original GTS solution after 60min of computation on 630 instances with uniformly random nodes.

We also recorded the ratio between the best solution on hand to the initial solution value every 100sec on these 630 instances while running the original GTS algorithm, and our modified algorithm. In Figure 41, we plot the averages of these ratios with respect to time for both versions. On average, the modified version remains as the winner compared to the original algorithm. So, we can conclude that on similar graphs, i.e., graphs with large number of uniformly distributed customers, creating the sparse graph considering the polar angles of the customers would generate better results in the same amount of time.

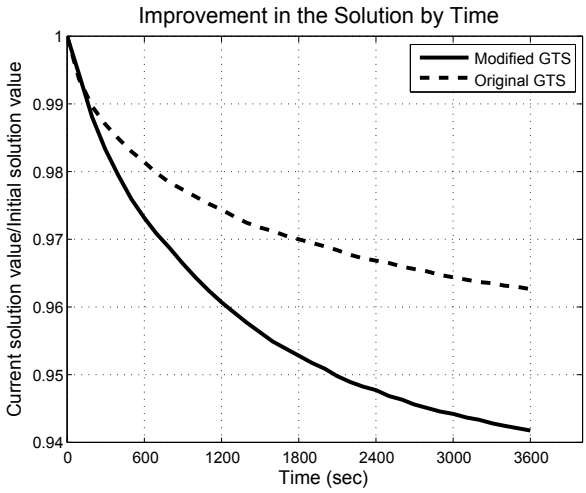


Figure 41: Comparison of improvement on the initial solution by time using the original and the modified GTS on random nodes.

3.6 Summary

With the increasing sizes of the supply chains, making good routing decisions are more difficult than before. Especially in the distribution systems where there are large number of customers and the set of customers to deliver items is changing from day to day, the same computational challenge has to be resolved everyday. When the schedule in a warehouse is tight, not enough computation time may be allocated even for fast heuristic methods. Therefore, using the conventional methods it may not possible to create time for computation without changing the order cutoff or truck dispatching.

Addressing this issue, in this chapter, we implemented Computation-Implementation Parallelization (CIP) policies on Time-Constraint Vehicle Loading and Routing Problem (TC-VLRP) both for Last-In-First-Out (LIFO) and random access truck loading. We empirically showed that it is possible to embed the computation into loading while attaining a target solution quality. By this means, the time we are saving can be used to allow a wider time window for the customers to place their orders or earlier truck dispatching.

Our results provide a basis for implementing CIP approach on other variants of routing problems, e.g., VRP with 2-, 3-dimensional loading constraints. As these integrated problems can provide better overall solutions, however they may be computationally too demanding for practical purposes.

We also modified the Granular Tabu Search (GTS) algorithm developed by Toth an Vigo [67]. We changed the way sparse graph is created; instead of choosing the short arcs, we choose the onces incident to the customers which are close to each other with respect to their polar angels made by the central depot. Modified sparse graph improved the solution by 2% on average.

3.7 Appendix: B Time Allocation Algorithms

As we noted in Section 3.4, if loading takes too long, our rules for dividing computation time among phases might need to be modified. In this section, we show the pseudo code for proportionally reducing the other phases' computation time. We first show some notation.

Notation:

- α: percentage of nodes we are fixing at each step,
- m: number of computation steps ($(m-1)$ computation steps are done in parallel with loading),
- $T^{'}$: Total available time for computation,
- $C^{'}[i]$: Computation time allocated for step i.

Algorithms 1, 2 and 3 show the pseudo code for each of our time allocation methods. In each case, if the loading time duration of the corresponding loading phase is greater than its initial allocated computation time $C[i]$, then we increase the computation time to $C^{'}[i] = \alpha n t_l$ for $i > 1$, and remove $C^{'}[i]$ from the total computation time for all computation phases $j < i$.

Algorithm 1 Determine computation time allocation for equal partitioning

$T^{'} \leftarrow T_0 + (m-1)\alpha nt_l$

for $i \leftarrow m$ **to** 1 **do**

  if $i = 1$ **then**

    $C^{'}[1] \leftarrow T^{'}$

    break

  end

  $C^{'}[i] \leftarrow \frac{T^{'}}{i}$

  if $C^{'}[i] < \alpha nt_l$ **then**

    $C^{'}[i] \leftarrow \alpha nt_l$

  end

  $T^{'} \leftarrow T^{'} - C'[i]$

end

Algorithm 2 Determine computation time allocation for linear partitioning

$T^{'} \leftarrow T_0 + (m-1)\alpha nt_l$

for $i \leftarrow 1$ **to** m **do**

  if $i = m$ **then**

    $C^{'}[1] \leftarrow T^{'}$

    break

  end

  SumL $\leftarrow \sum_{j=1}^{m}(1-(m-j)\alpha)$

  if $T^{'}(1-(m-i)\alpha)/SumL > \alpha nt_l$ **then**

    $C^{'}[m-i+1] \leftarrow T^{'}(1-(m-i)\alpha)/SumL$

  end

  else

    $C^{'}[m-i+1] \leftarrow \alpha nt_l$

  end

  $T^{'} \leftarrow T^{'} - C^{'}[m-i+1]$

end

Algorithm 3 Determine computation time allocation for quadratic partitioning

$T^{'} \leftarrow T_0 + (m-1)\alpha nt_l$

for $i \leftarrow 1$ **to** m **do**

  if $i = m$ **then**

    $C[1] \leftarrow T^{'}$

    break

  end

  $SumQ \leftarrow \sum_{j=i}^{m}(1-(m-j)\alpha)^2$

  if $T^{'}(1-(m-i)\alpha)^2/SumQ > \alpha nt_l$ **then**

    $C^{'}[m-i+1] \leftarrow T^{'}(1-(m-i)\alpha)^2/SumQ$

  end

  else

    $C^{'}[m-i+1] \leftarrow \alpha nt_l$

  end

  $T^{'} \leftarrow T^{'} - C'[m-i+1]$

end

Chapter IV

A DISTRIBUTION-FREE TSP TOUR LENGTH ESTIMATION MODEL FOR RANDOM GRAPHS

4.1 Introduction

TSP tour length estimation models are useful especially in the cases where we do not need to know what the exact tour is, yet we are interested in the optimal tour length. For example, some heuristics developed for the Location-Routing Problem (LRP) use iterative approximations of tour lengths to cut down on the computation time. Chien [19] showed that tour length estimations can be used to get good solutions for LRP. Nagy and Salhi [59] described how to use tour length estimation models as fast but approximate methods in LPR heuristics to evaluate trial moves. In applications where time is constrained, using tour length estimates can give the heuristics more time to find better solutions [60]. For example, time constraints can be an issue in mobile location routing problems where the facilities are set up temporarily. This can be due to high establishment cost, or highly changing demand; related application areas are cellular communication, humanitarian logistics, blood collection, and postal service in urban areas [39]. Since decisions in these applications are made repeatedly and time is limited, using good tour length estimation models to speed up the decision process to reach better solutions can provide benefits.

There are many tour length estimation models in the literature. However, they mostly study graphs where the nodes follow a known distribution, usually uniform. To enable the broader use of tour length estimation models, in this chapter we design a distribution-free model to estimate the tour length, that can be used with general node scatters where the distribution may not be known.

In their seminal paper, Beardwood et al. [7] showed that when the nodes are identically and independently distributed according to a probability density function f on a two-dimensional region R, the following holds:

$$\lim_{n\to\infty} T^* = \beta \int\int_R \sqrt{f_c(x,y)}dxdy\sqrt{n}, \tag{18}$$

where T^* is the optimal tour length, n is the number of the nodes, β is a constant, and f_c is the absolutely continuous part of f. Equation (18) becomes $T^* = \beta\sqrt{nA}$ for a uniform node dispersion on a region of area A. The value of β is unknown. Computational studies determined that the practical value of β is approximately 0.712 [22]. When the node dispersion is unknown, $\int\int_R \sqrt{f_c(x,y)}dxdy$ cannot be computed, so in such cases Beardwood et al.'s estimation model cannot be used. (Hereafter, we will call the factor $\beta \int\int_R \sqrt{f_c(x,y)}dxdy$ as β', where $\int\int_R \sqrt{f_c(x,y)}dxdy$ is computed for a distribution f on a unit area).

Eilon et al. [29] derived that the length of the optimal traveling salesman tour is proportional to a constant times $\sqrt{nA}$ for square regions and show that it applies to any arbitrary area A. Christofides and Eilon [20] assumed uniformity as well, and used small-size graphs for testing. Daganzo [26] proposed a heuristic-based tour length estimation formula for TSP. His model divides the area into strips of width w, and visits the nodes in the order that they are encountered along each strip. Following this approach, he found that the length of a tour is approximately $0.9n\sqrt{\delta}$, where δ is the density of the nodes. His focus in this paper was on uniform node dispersion. Daganzo [25] also developed a tour length estimation model specifically for the sort of tours one finds in CVRP. Robusté et al. [64] showed that Daganzo's model is suitable for cases when $7 < C < 1.5n/C$, where C is the maximum number of nodes that a vehicle can visit. Therefore, while the model is well suited for instances of CVRP, it is not designed for general TSP tours. Robusté et al. showed that Daganzo's model should be updated as follows if $n/C^2 > 1.5^{-1}$:

$$T^* \approx (0.9 + kn/C^2)\sqrt{nA}. \tag{19}$$

The value for constant k is suggested as 0.45 for squares and 0.55 for six-by-ten rectangles. However, in that formulation the coefficient of $\sqrt{nA}$ is greater than 0.9, which is significantly higher than the value of Beardwood et al.'s β, and leads to overestimate. Castillo [28] also derived an expression based on a tour construction heuristic. He derived this expression based on the uniform node distribution. Robusté et al. [63] provided formulas for elliptic regions for 3 different heuristics for VRP by regression. They also empirically verified Daganzo's and Robusté's formulas on small graphs. Platzman and Bartholdi [62] developed a heuristic based on space filling curves. The algorithm requires $O(nlogn)$ operations and $O(n)$ memory. When the distribution of the nodes is known, the model can be used directly to estimate the tour length, without needing to construct a tour. However, the tours constructed by this algorithm can be significantly longer than the optimal tour. A space filling curve heuristic was implemented by Bartholdi et al. [6] to construct tours for a non-profit organization, and generated solutions which are about 25% longer than optimum.

Most of the more-accurate estimation models focus on uniformly random node scatter or small graphs. To be able to estimate tour lengths on larger graphs even when we do not know the node dispersion without running a heuristic algorithm, in this chapter we develop a new tour length estimation model for graphs in which the node distribution is not restricted. Table 4 summarizes the need for our model. The literature contains TSP tour length estimates that are near the optimum, estimates that can be calculated quickly and estimated that do not require knowing the random node dispersion; but our new model is the only one with all three characteristics.

Table 4: Comparison of the models.

		Accurate (within 6% of optimal	Faster than constructing a tour	Does not require knowledge of node distribution
Asymptotic Models	Beardwood et al.	✓	✓	
Regression Models	Daganzo		✓	✓
	Chen		✓	✓
	Kwon et al.		✓	✓
Heuristics	Daganzo			✓
	$K-opt$	✓		✓
	Lin-Kernighan	✓		✓
	Platzman &Bartholdi			✓
This thesis	Çavdar & Sokol	✓	✓	✓

The rest of the chapter is organized as follows. In Section 4.2, we discuss the design of the model. In Section 4.3, we validate the model by testing it under different environments with small and large random graphs and identify the conditions when it works well. We also test the model using the statistics of the population distribution of the node coordinates rather than the exact locations of the nodes in Section 4.4. In section 4.5, we test the model on non-random graphs from the TSP library. In Section 4.6, we present some final remarks.

4.2 Design of the Estimation Model

Our goal is to develop a single model using regression to estimate optimal tour length of random graphs that will perform well with many node dispersion types and even when the distribution is unknown. For this purpose, we first specify the graph attributes that capture information about how the nodes are distributed on a graph.

4.2.1 Graph Attributes

We use some of the classical attributes from the literature, namely

- *Number of nodes*,
- *Area of the graph*.

In addition to these, we use the attributes listed below which have not been used in the literature before. These measures account for the effects of node dispersion on the optimal tour length.

- *Amount of node dispersion:* We use the standard deviation of the node coordinates in each dimension to collect information about the node dispersion.
- *Closeness to the boundary:* We measure closeness to the boundary by the average horizontal and vertical distance of nodes to the central axis.
- *Dispersion around the central axis:* We measure the dispersion of the nodes around the center by the standard deviation of their distances to the horizontal or vertical axis in the middle of the space.

The notation we will be using to denote the graph attributes is defined below.

Notation:

- n: number of nodes in the graph,
- l_x, l_y: lengths of the graph, i.e., the horizontal and the vertical dimensions of a rectangular graph,
- x_i, y_i: horizontal and vertical coordinates of each node i,
- $\overline{c}_x$, $\overline{c}_y$: average distance to the central horizontal and vertical axes,
- $stdev_x$, $stdev_y$: standard deviation of the horizontal and vertical coordinates,
- $cstdev_x$, $cstdev_y$: standard deviation of the absolute distances from the central horizontal and vertical axes,
- A: area of the graph,
- T: tour length.

4.2.2 Different Graph Types to Train the Model

Since the model is aimed to work well with many different node dispersions, we generated six types of rectangular graphs to train the model. In all these graphs, the first 4 nodes are placed on the corners to ensure that the total area within the convex hull is the same.

1. *Uniform Graph:* The coordinates of each node are uniformly distributed in each dimension independently on the intervals $[0, l_x]$, and $[0, l_y]$.

2. *Triangular Graph:* The horizontal and vertical node coordinates are randomly generated by independent symmetric triangular distributions with parameters $(0, \frac{l_x}{2}, l_x)$ and $(0, \frac{l_y}{2}, l_y)$.

3. *Squeezed Graph:* In this graph, the nodes are denser around the right upper corner and less dense around the left lower corner.

   To obtain the required node dispersion, first we generate uniformly random nodes on the plane. Then, we only keep the ones meeting the following acceptance condition: A random variable *accept* is generated by $U(0, l_x \cdot l_y)$. If the generated node is close enough to upper right corner, i.e., $x \cdot y$ is larger than *accept*, we keep the node. Otherwise, we discard it and generate a new one. The pseudo code to generate each node from this dispersion is given below.

   repeat

   $x \leftarrow U(0, l_x)$

   $y \leftarrow U(0, l_y)$

   $accept \leftarrow U(0, l_x \cdot l_y)$

   until $accept < x \cdot y$

4. *x-axis: Uniform, y-axis: Triangular*: This is a hybrid graph. Horizontal coordinates are generated by $U(0, l_x)$ and vertical coordinates are generated by $T(0, \frac{l_y}{2}, l_y)$.

5. *x-axis: Triangular, y-axis: Squeezed:* This is another hybrid graph where the horizontal coordinates are generated by $T(0, \frac{l_x}{2}, l_x)$, and vertical coordinates are squeezed around the ceiling as described below.

   repeat

   $y \leftarrow U(0, l_y)$

   $accept \leftarrow U(0, l_y)$

   until $accept < y$

6. *x-Central Graph:* Vertical coordinates of the nodes are denser around the boundary, but horizontal coordinates are denser around the center. Each node is generated as described below.

   repeat

   $x \leftarrow U(0, l_x)$

   $y \leftarrow U(0, l_y)$

   $accept \leftarrow U(0, 1)$

   until $accept < (1 - \frac{|x - \frac{l_x}{2}|}{\frac{l_x}{2}})(\frac{|y - \frac{l_y}{2}|}{\frac{l_y}{2}})$

Each of these six graphs is illustrated by one example with 1,000 nodes in Figures 42, 43, 44, 45, 46 and 47.

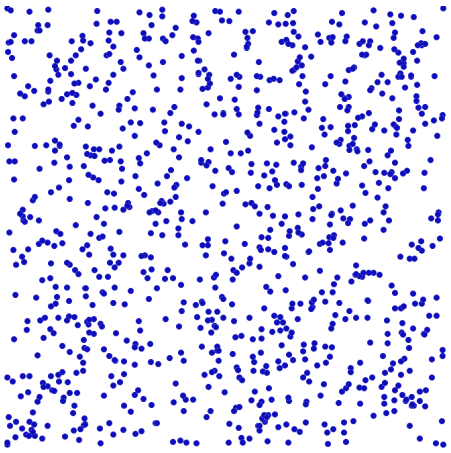


Figure 42: Uniform dispersion.

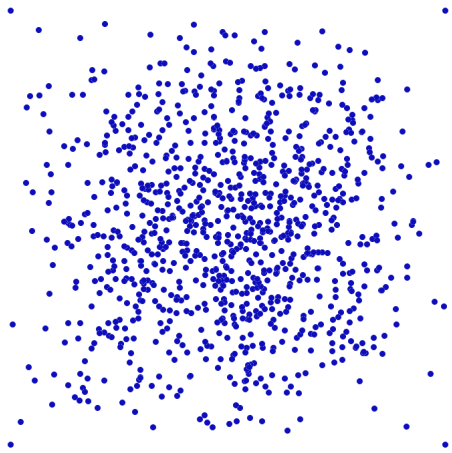


Figure 43: Triangular dispersion.

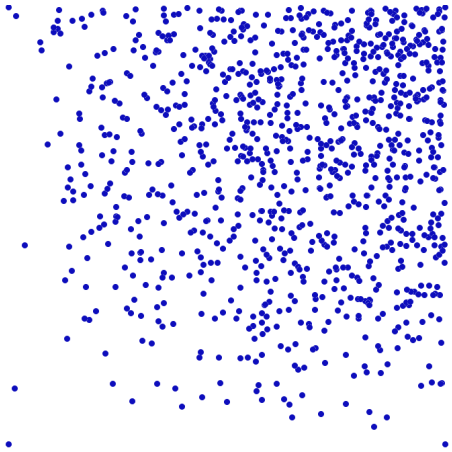


Figure 44: Squeezed dispersion.

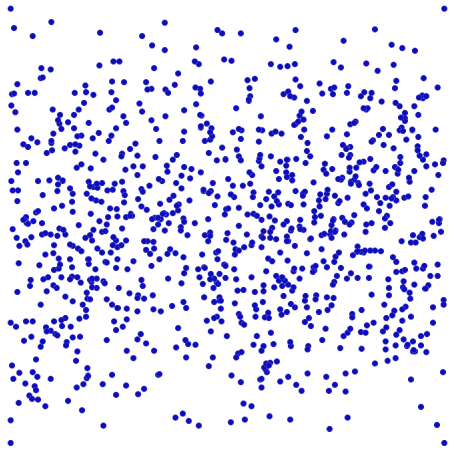


Figure 45: x: Uniform y: Triangular.

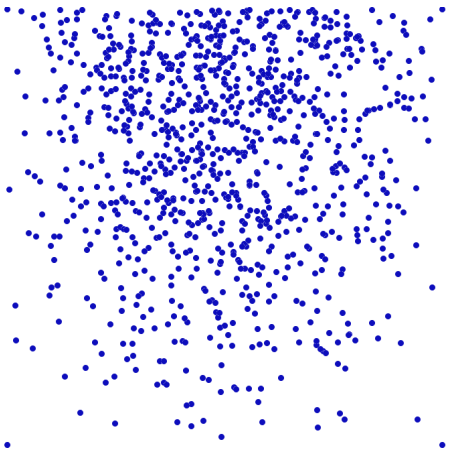


Figure 46: x: Triangular y: Squeezed.

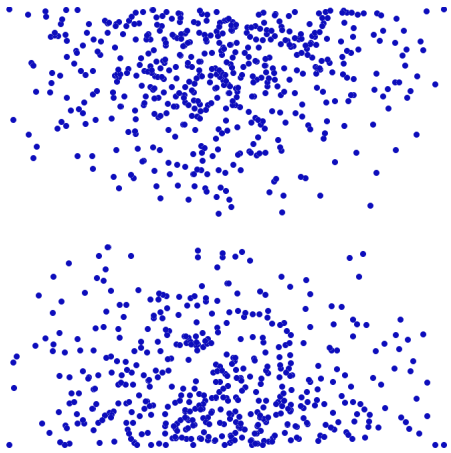


Figure 47: x: Central y: Boundary.

To train the model, 400 graphs were generated for the aforementioned six graph types. The parameter combinations to generate the graphs are obtained by crossing the elements from the following sets.

- $n \in \{3000, 4000, 5000, 5200, 5400, \ldots, 6600, 7000, 8000\}$,
- $l_x \in \{250, 300, 400, 500, 600, 800, 1000, 1200, 1600\}$,
- $l_y \in \{\frac{250000}{l_x}, \frac{360000}{l_x}, \frac{640000}{l_x}\}$.

Since our graphs are large, finding optimal solutions to all of them is impractical. So, we used the Lin-Kernighan (LK) implementation by Applegate et al. [44] to find tour length estimates to train the model. This algorithm is known to produce results within 1% of optimal [1]. Using regression, we discovered the following model for LK tour length estimation:

$$T \approx 2.791\sqrt{n(cstdev_x cstdev_y)} + 0.2669\sqrt{n(stdev_x stdev_y)\frac{A}{\bar{c}_x \bar{c}_y}} \tag{20}$$

The model has a high predicting power for the training data implied by R^2 =0.9956, and the other statistics shown in Table 5 indicate that the coefficients are significant and have low relative errors. Figure 48 demonstrates the relation between the tour length found by the LK algorithm and the estimated tour length by the model in equation (20). Figure 49 presents the same comparison for the estimation model proposed by Beardwood et al. in equation (18); β' is calculated for each dispersion.

However, finding the suitable β' is not possible when the node dispersion is unknown. In such cases, as seen in Figure 48, our model still provides reliable estimates capturing the differences in the dispersions correctly, but Figure 50 shows that using the wrong β' when the distribution is unknown can lead to a larger estimation errors. Thus, our model is especially useful for graphs with unknown dispersion.

Table 5: Statistics for the model in equation (20).

Coefficient	Value	Standard Error	t value	Pr(>| t |)
$\sqrt{n(cstdev_x cstdev_y)}$	2.791	0.02714	102.846	$< 2e$-16
$\sqrt{n(stdev_x stdev_y)\frac{A}{\bar{c}_x \bar{c}_y}}$	0.2669	0.003508	76.073	$< 2e$-16

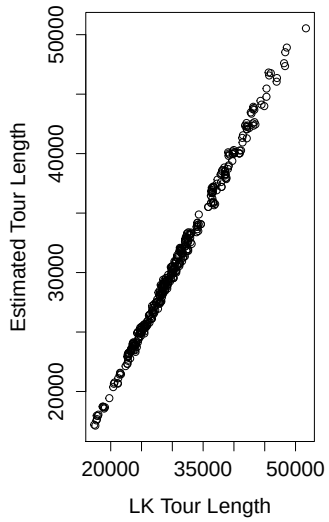


Figure 48: Lin-Kernighan tour length versus estimation by our model in equation (20).

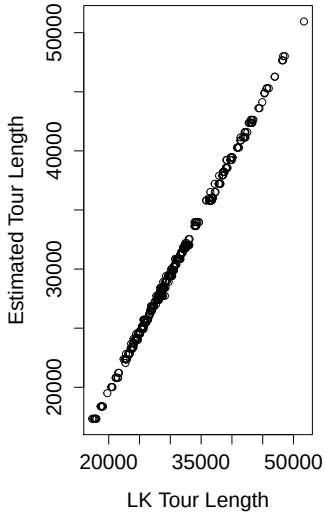


Figure 49: Lin-Kernighan tour length versus estimation by the model in equation (18) when β' is calculated for each dispersion.

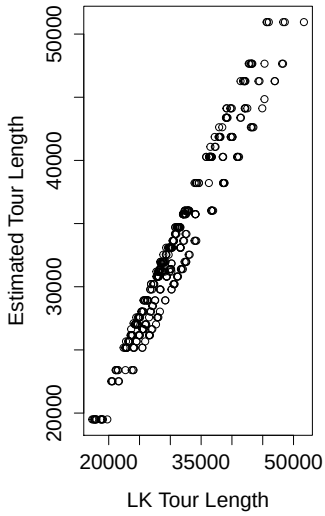


Figure 50: Lin-Kernighan tour length versus estimation by the model in equation (18) when we do not know the correct value of β', and use the uniform β'=0.712 for all dispersions .

4.3 Testing on Random Graphs

To measure the performance of the tour length estimation model, we tested it on a more varied sets of graphs than we used for training. The first test set has the same node distribution and the same parameters as the training set, but the graphs are different. In the second test set, we generate graphs with the same dispersions but with different parameters. In the third set of tests, we use different node dispersions than the ones we used in training. Then, we test the model on nonrectangular graphs. Finally, we test the model on symmetric TSP instances from the literature [43, 45], many of which are not randomly generated.

As before, the optimal tour lengths are estimated by the Lin-Kernighan (LK) implementation in [44]. As the performance measure we use the ratio of our model's estimated tour length (E) to the LK tour length (T).

The sets of test graphs were generated as follows:

1. [$G1$] We generated graphs for the six node dispersions we used to train the model, using the same area and number of nodes, but different random seeds.

2. The goal of the second test set is to measure the estimation model's robustness to differences in the number of nodes and the elongation of the graphs generated by using the same six training dispersions:

   (a) [$G2.1$] Graphs having 40% more nodes than the training data in the same area,

   (b) [$G2.2$] Graphs with the same number of nodes as the training data, but 20% longer and 40% wider,

   (c) [$G2.3$] Graphs having between 20,000 and 30,000 nodes in a rectangular area where the length and the width are uniformly and independently randomly distributed between 1,000 and 2,000,

(d) [$G2.4$] Graphs having between 100,000 and 200,000 nodes in a rectangular area where the length and the width are uniformly and independently randomly distributed between 1,000 and 2,000, also using the six training dispersions.

In this group of tests, the density of the graphs, n/A, comes from a wider range than in the training data.

3. We also generated three new graph types to test the model on different node dispersions from those it was trained on:

   [$G3.1$] Rectangular graphs where the nodes are denser around the corners, and there is a cavity in the center.

   The pseudo code to generate this type of graph is given below.

   repeat

   $x \leftarrow U(0, l_x)$

   $y \leftarrow U(0, l_y)$

   $accept \leftarrow U(0, 1)$

   until $accept < \frac{|x-\frac{l_x}{2}|}{\frac{l_x}{2}} \frac{|y-\frac{l_y}{2}|}{\frac{l_y}{2}}$

   An example graph with 1,000 nodes can be seen in Figure 51.

   [$G3.2$], [$G3.3$] Two other graph types were generated using truncated exponential distributions. To create the horizontal and vertical coordinates of the nodes, we used truncated exponential distributions in (0,1) with rate 1 for [$G3.2$] and rate 2 for [$G3.3$]. Multiplying them by l_x and l_y, we obtained the node coordinates. The pseudo code used to generate this dispersion type is given below.

   $rate \leftarrow 1$ or 2

   $a \leftarrow -\log(U(0, 1))/rate$

$b \leftarrow -\log(U(0,1))/rate$

$x \leftarrow (a - \lfloor a \rfloor) \cdot l_x$

$y \leftarrow (b - \lfloor b \rfloor) \cdot l_y$

Figure 52 presents an example graph generated for $[G3.2]$ and Figure 53 presents an example graph generated for $[G3.3]$. The parameter sets to generate these three new graphs for test groups $[G3.1]$, $[G3.2]$, $[G3.3]$ are as follows:

- $n \in \{5000, 5500, \ldots, 100000\}$,
- $l_x \in \{250, 500, 1000\}$,
- $l_y \in \frac{100{,}000}{l_x}$.

This data set also allows us to test a broader range of densities (n/A) than the training group.

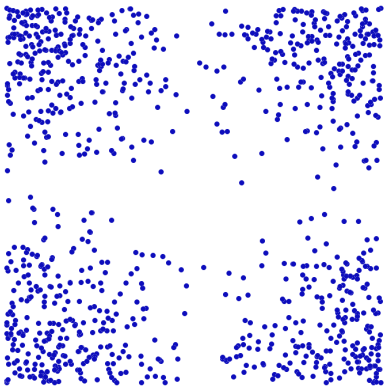


Figure 51: Cavity dispersion.

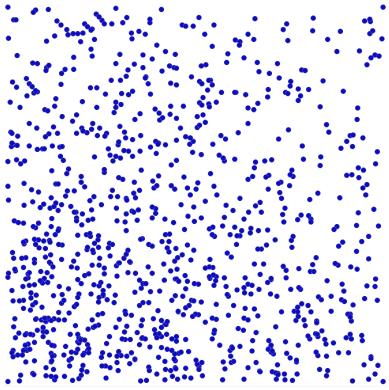


Figure 52: TExp(1).

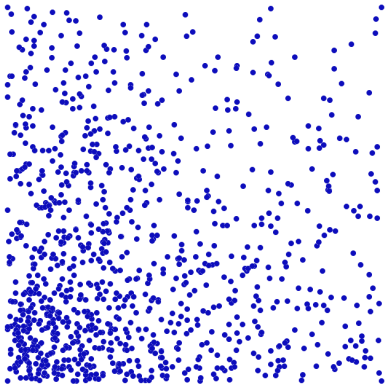


Figure 53: TExp(2).

Having created all these instances for eight different test settings, we tested our model together with six other tour length estimation models in the literature. Table 6 shows the other models that we compare with. All the estimators assume Euclidean distances. D denotes the average distance from the depot in these models. δ denotes the density of the nodes. While training our model, we did not specify the location of the depot. For the test sets, we assume that the node at the origin is the depot, because Chien trained his model assuming that the depot is at the origin, and Kwon et al. trained their models both for depot in the center and at the origin. For completeness, we also ran the same

tests with the depot in the center. The results were not significantly different, so they are not reported in detail. R denotes the ratio of the length to width, assuming that the longer dimension is the length. A' is the area of the smallest rectangle covering only the customers, i.e., not the depot. Since our test graphs have a large number of nodes, one node at each corner of the rectangle and they are dense around the origin, we use the whole graph area A as A'.

Daganzo's CVRP-based model [25] we are comparing with was developed for multiple vehicle routing problems. Others are trained on smaller size instances, and mostly uniform node dispersion. Although these models perform well on the types of instances they are designed for, they should not be expected to perform well on the test cases in this chapter. However, we still include them in the test results for completeness.

Table 6: Estimation models in the literature we compare our model with.

Source	Model
Beardwoord et al. [7]	$\beta\sqrt{nA}$
Chien [18]	$2D + 0.69\sqrt{A'(n-1)}$
Daganzo [25] (CVRP)	$2D + 0.57\sqrt{A'(n-1)}$
Daganzo [26] (TSP)	$0.9n/\sqrt{\delta}$
Kwon et al. [50] (1)	$(0.8326 - 0.0011n + 1.1147R/n)\sqrt{nA}$
Kwon et al. [50] (2)	$(0.7754 - 0.0008n + 0.9027R/n)\sqrt{nA} + 0.4147\mathrm{D}$

In Table 7, we report the test results in terms of estimated tour length (E) divided by the tour length (T) found by the LK implementation [44]. Daganzo's CVRP-based model [25] underestimates the tour length according to average performance, i.e., the average E/T is between 0.83 and 0.92 for this model. Chien's [18] model, whose only difference from Daganzo's [25] model is to use 0.69 instead of 0.57, overestimates on the average for all test groups except for

[$G3.2$]. This indicates that a constant between 0.57 and 0.69 as the coefficient of the term $\sqrt{A'(n-1)}$ would generate more accurate estimates. For this reason, instead of using the updated version of Daganzo's model by Robusté et al., which is $(0.9+kn/C^2)\sqrt{nA}$, we used the original model as developed by Daganzo, even though $7 < C < 1.5n/C$ is not satisfied. Daganzo's [26] heuristic-based model provides overestimates. We have not re-estimated the parameters of the models that we compared with. Kwon et al.'s [50] models returned either negative or very low values when the number of nodes in a graph is large, due to the negative coefficient of n in the models. Therefore, we did not include those in our comparison.

Table 7: Comparison of the performance of the estimation models on different testing settings. The numbers on the table are the average, minimum, maximum and the standard deviation of estimated tour length divided by LK tour length for the corresponding models.

	[*G1*]				[*G2.1*]			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.00	0.97	1.03	0.01	1.01	0.97	1.03	0.01
Daganzo (CVRP)	0.90	0.81	0.96	0.04	0.90	0.81	0.95	0.04
Daganzo (TSP)	1.36	1.23	1.43	0.06	1.36	1.23	1.43	0.06
Chien	1.08	0.98	1.15	0.05	1.08	0.97	1.14	0.05
Beardwood et al.	0.98	0.95	1.00	0.01	0.99	0.97	1.00	0.01
	[*G2.2*]				[*G2.3*]			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.00	0.97	1.03	0.00	1.01	0.99	1.03	0.00
Daganzo (CVRP)	0.90	0.81	0.97	0.04	0.88	0.81	0.92	0.04
Daganzo (TSP)	1.36	1.23	1.43	0.06	1.37	1.25	1.42	0.06
Chien	1.08	0.98	1.16	0.05	1.07	0.97	1.10	0.05
Beardwood et al.	0.98	0.95	1.01	0.01	0.99	0.99	1.00	0.00
	[*G2.4*]				[*G3.1*]			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.01	0.99	1.03	0.01	0.96	0.94	0.97	0.00
Daganzo (CVRP)	0.88	0.80	0.91	0.04	0.92	0.91	0.96	0.01
Daganzo (TSP)	1.38	1.26	1.42	0.06	1.41	1.38	1.42	0.01
Chien	1.06	0.97	1.10	0.05	1.10	1.09	1.15	0.01
Beardwood et al.	1.00	0.99	1.00	0.00	0.99	0.97	1.00	0.00
	[*G3.2*]				[*G3.3*]			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	0.99	0.98	1.00	0.00	1.00	0.99	1.01	0.00
Daganzo (CVRP)	0.83	0.82	0.86	0.01	0.87	0.86	0.90	0.01
Daganzo (TSP)	1.28	1.26	1.28	0.00	1.36	1.33	1.36	0.01
Chien	1.00	0.99	1.03	0.01	1.05	1.05	1.08	0.01
Beardwood et al.	0.99	0.98	1.00	0.00	0.99	0.97	0.99	0.00

In test groups $[G1]$, $[G2.1]$, $[G2.2]$, $[G2.3]$, and $[G2.4]$, the graphs have six different node dispersions with different elongation factors. When we calculate the standard deviation of the errors in the tour length estimations, we observe that the models in the literature other than Beardwood et al. [7] have a higher standard deviation compared to our model, between 4% and 5% as opposed to 1%. Overall, these results are not meant to imply the deficiency in the models from the literature, since they are only designed on uniform graphs; instead, Table 7 shows the necessity for our model for more general graphs where the distribution may not be known.

$[G3.1]$ is the only test group where our model fails to return average E/T inside the interval [0.99, 1.01]. Although our model gives the closest estimates to the tour length, it underestimates. Our conjecture is that the statistics we are collecting are not well reflective of the graph attributes, because the cavity dispersion (Figure 51) looks like a combination of four subgraphs, each similar to the squeezed dispersion (Figure 44). So, we ran an additional test. We divided the cavity graph into four subgraphs, estimated the tour length for each and added these estimates to find the tour length estimation for the whole graph. We used the same parameters as before, except the maximum number of nodes was 50,000 instead of 100,000; otherwise, the subgraphs merge and the whole graph cannot be divided into independent subsets properly.

Since other models do not require such an adjustment, we made it for only our estimation model. The summary of this test can be seen in Table 8. Considering the average ratio between the estimated tour length and LK tour length, the performance of our model is improved, i.e., the estimation/tour length ratio is now 1.02, at the expense of a slight increase in the standard deviation in the errors.

Table 8: Average, minimum, maximum and the standard deviation of the estimated tour length divided by the LK tour length for the subgraph estimation method for cavity graphs. The final estimate is obtained by adding the estimation for the four subgraphs in the cavity dispersion graph.

Model	Average	Min	Max	Stdev
General	1.02	0.99	1.03	0.01

Our model tends to underestimate when the number of nodes in the graph is small. The plot in Figure 54 shows the relation between the number of nodes in a graph and the E/T ratio. The model underestimates when there are fewer than 1,000 nodes in the graph, but after this point the estimation is proper and stationary. The E/T factor can be expressed as an exponential function of number of nodes as in equation (21).

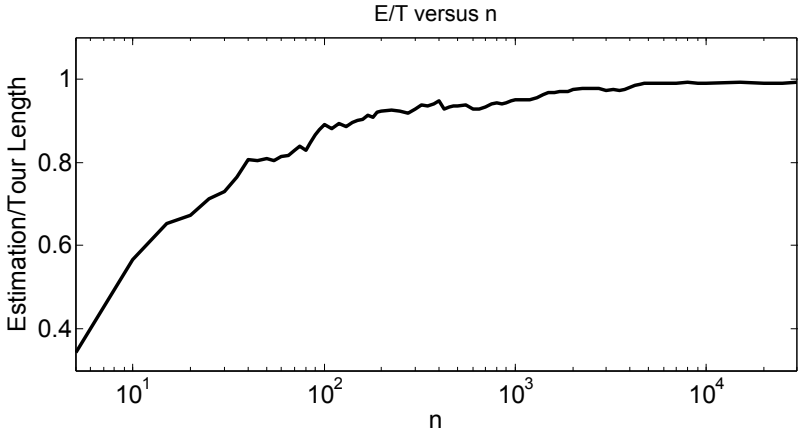


Figure 54: Relation between the number of nodes and Estimation/Tour Length ratio.

$$\frac{E}{T} = 0.9325e^{0.00005298n} - 0.2972e^{-0.01452n} \tag{21}$$

The R^2 of this fit is 0.9867. We tested the model on graphs with small number of nodes, correcting estimation according to equation (21). The parameter set

used in this test is as follows:

- $n \in \{100, 125, \ldots, 975\}$,
- $l_x = \{500, 1000\}$,
- $l_y = \frac{1{,}000{,}000}{l_x}$.

The comparative results are presented in Table 9. Based on the average E/T, our model performs better. Kwon et al.'s [50] second model also returns average E/T ratios that are close to 1, but the deviations in the errors are higher. Chien [18] and Kwon et al.'s [50] first model do not perform well on this test. More detailed comparison data can be seen in Table 9. A similar correction factor to that we used for our model could have been used for the others as well. However, to the best of our knowledge no such factors are available for them.

Table 9: Average, minimum, maximum and the standard deviation of the estimated tour length divided by the LK tour length for small-size graphs.

Model	Average	Min	Max	Stdev
General with correction	0.99	0.92	1.08	0.02
Daganzo (CVRP)	0.96	0.82	1.12	0.06
Daganzo (TSP)	1.27	1.05	1.44	0.08
Chien	1.13	0.98	1.28	0.07
Beardwood et al.	1.00	0.86	1.01	0.01
Kwon et al. (1)	0.33	-0.41	1.03	0.42
Kwon et al. (2)	1.03	0.90	1.18	0.06

4. [$G4$] So far, in all the test groups we measured the performance of the tour length estimation model on rectangular areas. To observe the model's robustness to distortion in the graph shape, we tested the estimation model on graphs with different shapes.

   In this test group, we first generated m nodes randomly on a rectangle with

length l_x and width l_y, and determined the convex hull of these initial nodes. Then, the remaining $n - m$ nodes are generated in this convex hull. For large enough m the graphs look like rectangles, but for small m the graph shapes may be very different from what we trained our model on. For small m, since the graphs are not necessarily rectangles and do not lie parallel to the x-axis, we rotate the graph so that the ratio of the area of the convex hull to the area of the smallest rectangle covering all the nodes and lying parallel to the x-axis is maximum. Freeman and Shapira [31] showed that the rectangle of minimum area enclosing a convex polygon has a side collinear with one of the edges of the polygon. Having created the graphs, we first determined the optimal rotation and calculated the statistics we used in the estimation using the rotated graph. The parameter sets for this test group are as follows:

- $n \in \{2000, 4000, 6000, 8000, 10000\}$,
- $m \in \{5, 10, 15, \ldots, 50, 70, 100\}$,
- l_x, l_y =1,000.

For each m, one uniformly randomly generated graph with 2,000 nodes is presented in Figures 55 to 66 below. Those figures show the change in the shape of graphs as the number of initial nodes m generated to create the convex hull increases. For each different m, 120 graphs were generated using the graphs types shown in Figures 42, 43, 44, 45, 46, 47, 52 and 53. The graphs are rotated when necessary, as above. Detailed comparison data is provided in Table 10.

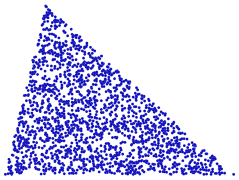


Figure 55: m=5

Figure 56: m=10

Figure 57: m=15

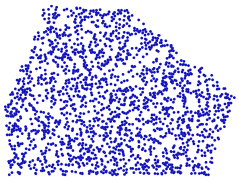


Figure 58: m=20

Figure 59: m=25

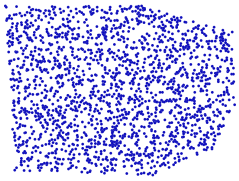


Figure 60: m=30

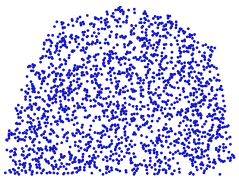


Figure 61: m=35

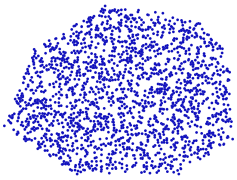


Figure 62: m=40

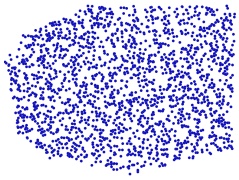


Figure 63: m=45

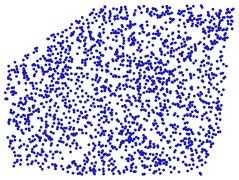


Figure 64: m=50

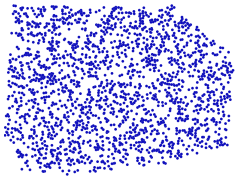


Figure 65: m=70

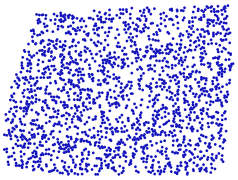


Figure 66: m=100

Table 10: Comparison of the performance of the estimation models on non - rectangular graphs. The numbers on the table are the average, minimum, maximum and the standard deviation of estimated tour length divided by LK tour length for the corresponding models.

	m=5				m=10			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.06	0.98	1.12	0.03	1.03	0.97	1.09	0.03
Daganzo (CVRP)	0.87	0.81	0.95	0.03	0.87	0.81	0.94	0.03
Daganzo (TSP)	1.31	1.21	1.39	0.05	1.32	1.23	1.40	0.05
Chien	1.05	0.98	1.13	0.04	1.04	0.97	1.12	0.04
Beardwood et al.	0.96	0.89	1.00	0.02	0.96	0.91	1.00	0.02
	m=15				m=20			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.02	0.98	1.09	0.02	1.01	0.97	1.09	0.02
Daganzo (CVRP)	0.87	0.81	0.93	0.03	0.86	0.81	0.92	0.03
Daganzo (TSP)	1.32	1.22	1. 40	0.04	1.32	1.24	1.39	0.04
Chien	1.04	0.97	1.11	0.04	1.04	0.98	1.10	0.04
Beardwood et al.	0.96	0.92	1.00	0.02	0.96	0.91	0.99	0.02
	m=25				m=30			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.01	0.98	1.06	0.02	1.01	0.96	1.05	0.02
Daganzo (CVRP)	0.87	0.81	0.92	0.03	0.87	0.81	0.93	0.03
Daganzo (TSP)	1.32	1.22	1.40	0.05	1.32	1.23	1.40	0.05
Chien	1.04	0.98	1.10	0.04	1.04	0.97	1.11	0.04
Beardwood et al.	0.97	0.92	0.99	0.02	0.97	0.91	0.99	0.02
	m=35				m=40			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.01	0.97	1.05	0.01	1.01	0.97	1.08	0.02
Daganzo (CVRP)	0.87	0.81	0.93	0.03	0.87	0.81	0.93	0.03
Dagaonzo (TSP)	1.32	1.23	1.41	0.05	1.32	1.22	1.40	0.05
Chien	1.04	0.97	1.12	0.04	1.05	0.98	1.11	0.04
Beardwood et al.	0.97	0.91	0.99	0.02	0.97	0.93	0.99	0.02
	m=45				m=50			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.00	0.98	1.04	0.02	1.00	0.97	1.04	0.01
Daganzo (CVRP)	0.87	0.81	0.93	0.03	0.87	0.81	0.92	0.03
Daganzo (TSP)	1.32	1.23	1.41	0.05	1.32	1.22	1.40	0.05
Chien	1.05	0.98	1.12	0.04	1.04	0.98	1.10	0.04
Beardwood et al.	0.97	0.92	0.99	0.01	0.97	0.91	0.99	0.02
	m=70				m=100			
Model	Average	Min	Max	Stdev	Average	Min	Max	Stdev
General	1.00	0.96	1.02	0.01	1.00	0.96	1.03	0.01
Daganzo (CVRP)	0.87	0.81	0.92	0.03	0.87	0.81	0.92	0.03
Daganzo (TSP)	1.32	1.22	1.41	0.05	1.32	1.22	1.39	0.05
Chien	1.05	0.98	1.11	0.04	1.05	0.97	1.10	0.04
Beardwood et al.	0.97	0.92	0.99	0.02	0.97	0.92	1.00	0.01

Performance of our estimation model improves with an increase in m as expected. However, as it is seen in Table 10, even when a graph does not have a rectangular shape, acceptable estimations can be obtained.

In this section we presented several test results to measure the performance of our estimation model in different settings. We observed that the model introduced in this chapter can provide reliable estimates on the graph types it was trained for, and it also returns reliable estimates for node dispersions and graph dimensions different from the training data. The results we presented are the summaries of the ratios between the estimated tour length and the length of the tour found by LK implementation [44]. An estimated comparison between our results and the optimal tour length can be obtained by multiplying the numbers in the summary tables by (1+average deviation of LK from optimal), which does not change the relative comparison between our estimation model and the models in the literature.

4.4 Comparison to Beardwood et al.'s (1959) β Coefficient

In the previous section, the computational results show that our model has a high predictive power. In this section, we will show that it can also be used to estimate the coefficient of Beardwood et al.'s model, for example, in a case where the integration is difficult. Of course, when the dispersion is unknown, we can try to fit a distribution on the node coordinates, and use it to estimate β'. However, distribution fitting also takes time, and there will be an error in distribution fitting in addition to the estimation error.

To make comparisons, we first computed β' for the other node dispersions we use in this chapter. For example, it is a straightforward integration to compute $\int\int \sqrt{f_c(x,y)}dxdy$ for the Triangular node dispersion on a unit square as 0.889,

so $\beta' = 0.712 * 0.889 = 0.633$. Beardwood et al.'s estimation model for the Triangular node dispersion is therefore $T^* \approx 0.633\sqrt{nA}$.

In our estimation model in (20), if we replace sample statistics *cstdevx*, *stdevy*, *stdevx*, *stdevy*, $\bar{c}_x$ and $\bar{c}_y$ by the population statistics (when we know the node dispersion), it reduces to $T \approx \kappa\sqrt{nA}$. This is the same form as Beardwood et al.'s model. So, the values of κ should match with the values of β' as described above. For example, for the Triangular node dispersion, by plugging the population statistics into (20), we obtain $T \approx 0.656\sqrt{nA}$. To show that the κ's for each node dispersion are actually a good estimates for β', we computed them for the other node dispersions we used in this chapter. Table 11 displays the β' and κ values for the corresponding dispersions. As it is seen, κ's are within 4% of the β''s.

Table 11: Comparison of κ and β''s for different node dispersions. Numbers in the parenthesis are the β''s.

Uniform	Triangular	Squeezed
0.711 (0.712)	0.656 (0.633)	0.654 (0.633)
x: Uniform y: Triangular	x: Triangular y: Squeezed	x: Central y: Boundary
0.681 (0.671)	0.651 (0.633)	0.633 (0.633)
Cavity	TExp(1)	TExp(2)
0.612 (0.633)	0.699 (0.698)	0.667 (0.658)

We also repeated the tests in the previous section on the same graphs using the estimate $T \approx \kappa\sqrt{nA}$ instead of exact node coordinates. The results are presented in Table 12. These results supports the claim that the model can also estimate the β' values when we cannot compute them.

Table 12: Average, minimum, maximum and the standard deviation of the estimated tour length divided by the LK tour length when we use the statistics of the population distribution rather the statistics of the actual node coordinates in our estimation model.

Test	Average	Min	Max	Stdev
$[G1]$	1.00	0.97	1.03	0.01
$[G2.1]$	1.01	0.97	1.03	0.01
$[G2.2]$	1.00	0.97	1.03	0.01
$[G2.3]$	1.01	0.99	1.03	0.01
$[G2.4]$	1.01	0.99	1.03	0.01
$[G3.1]$	0.96	0.94	0.97	0.00
$[G3.2]$	1.00	0.98	1.00	0.00
$[G3.3]$	1.01	0.98	1.01	0.01
Small Graphs	0.92	0.65	1.01	0.07
$[G3.1]$ (*Divided into subgraphs*)	1.03	1.00	1.03	0.00
$[G4]$ $(m = 100)$	1.01	0.96	1.05	0.02

4.5 Testing on Non-Random Graphs

Having shown that our model works well on random graphs for many node dispersions, we decided to test it on instances from the TSP Library [43] and National TSP instances [45], most of which are not random, to determine the model's performance on non-random graphs. For graphs having fewer than 1,000 nodes, the estimation is corrected according to equation (21). In these tests, we used the area of the convex hull of the nodes as A. Also, since these graphs do not have a predefined depot, we used the boundary node with the smallest total distance to the other nodes as the depot. Since not all these graphs are rectangular in shape, in order to decrease the effect of mis-rotation of the graphs on the estimation models' performance, we rotated the graphs so that they have the optimal position, i.e., the ratio of area of the convex hull to the area of the minimum area-rectangle covering all the nodes and

lying parallel to the x-axis is maximum, as before.

Many of these graphs have different attributes from the ones we trained and tested on, specifically because they are not randomly generated. These different attributes include:

(I) Graphs that are composed of distinct subgraphs, e.g., instance p152 in Figure 67,

(II) Graphs where nodes are placed on a grid, e.g., instance u1432 in Figure 68,

(III) Graphs with non-rectangular shapes, e.g., instance brd14051 in Figure 69,

(IV) Graphs where the convex hull has a significant empty space inside its convex hull, e.g., Yemen in Figure 70.

Figure 67: p152.

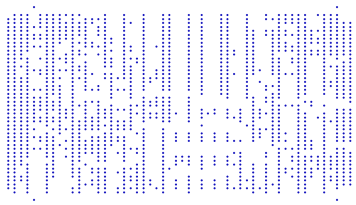


Figure 68: u1432.

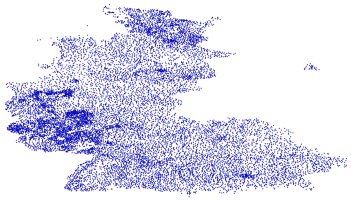


Figure 69: brd14051.

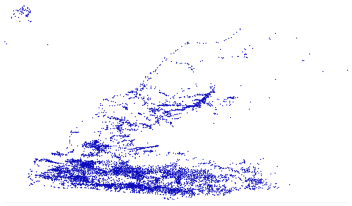


Figure 70: Yemen.

The summary of the estimation results of our model and the two other models in the literature are presented in Table 13. In those tables, the graphs are grouped according to their common attributes. Some graphs repeat in more than one group, e.g., p152 in Figure 67 has both attributes (I) and (IV). These results reveal that none of the estimation models provides a reasonable estimation for graphs having at least one of the attributes (I), (II), (III), and (IV). On average they overestimate and

the standard deviations are very high. However, when graphs do not have any of these attributes, the estimation results improve. Our model and Chien [18] perform better compared to Daganzo [25]. Beardwood et al. [7] is not reported because their β value cannot be calculated for these graphs.

Table 13: Comparison of the performance of the estimation models on TSP instances in the literature. The numbers on the table are the average, minimum, maximum and the standard deviation of estimated tour length divided by LK tour length for the corresponding models.

Graph Attribute	Model	Average	Min	Max	Stdev
(I)	General	1.48	0.89	2.84	0.48
	Daganzo (CVRP)	1.36	0.85	2.46	0.43
	Daganzo (TSP)	1.89	1.09	3.73	0.69
	Chien	1.61	0.99	2.96	0.52
(II)	General	1.33	0.73	2.84	0.43
	Daganzo (CVRP)	1.18	0.63	2.46	0.41
	Daganzo (TSP)	1.66	0.94	3.73	0.62
	Chien	1.40	0.75	2.96	0.49
(III)	General	1.37	0.98	2.84	0.46
	Daganzo (CVRP)	1.22	0.78	2.46	0.49
	Daganzo (TSP)	1.72	0.83	3.73	0.82
	Chien	1.45	0.89	2.96	0.60
(IV)	General	1.46	0.95	2.84	0.45
	Daganzo (CVRP)	1.33	0.77	2.46	0.47
	Daganzo (TSP)	1.98	1.11	3.73	0.72
	Chien	1.60	0.91	2.96	0.57
None	General	0.99	0.86	1.09	0.06
	Daganzo (CVRP)	0.85	0.75	0.94	0.06
	Daganzo (TSP)	1.11	0.87	1.33	0.10
	Chien	1.00	0.87	1.12	0.07

4.6 Summary

Motivated by the lack of a model in the literature to estimate the TSP tour length for unknown node distributions, in this chapter we introduced a new distribution-free estimation model. We trained and tested our regression model on graphs which

are specifically designed to reflect several node dispersions in addition to uniform randomness. We tested the model on graphs up to 200,000 nodes and showed that our model produces reliable estimates even for dispersions different from what it was trained on.

We also demonstrated that our model is similar to Beardwood et al.'s model when we replace the sample statistics with the population statistics. This explains why these two models perform similarly to each other when the node dispersion is known, and supports our model's accuracy for the cases when the node dispersion is unknown, and none of the models in the literature works well.

Chapter V

CONCLUSIONS

5.1 Summary of Results

In this dissertation, we focus on developing, testing and analyzing methods of Computation-Implementation Parallelization (CIP) for certain time-sensitive applications. The main goal is being able to complete the tasks in a shorter amount of time.

Traditionally, heavy computational requirements are managed either using a faster heuristic solution method (which generally produces lower quality of solutions) and/or imposing a time limit on the computation time. This forces us to trade the solution quality for a shorter computation time. As an alternative, we introduced CIP approaches to special cases of TSP and VRP. We computationally showed that it is possible to reach the target solution quality by embedding almost all the computation into the solution-implementation.

While working on the CIP concept, some tangential questions came up; we made a geometric analysis of the 2*opt* heuristic for TSP, developed a TSP tour length estimation model which can be used independent of the node dispersion, and improved the performance of Toth and Vigo's [67] GTS algorithm for VRPs on uniformly distributed nodes. In the rest of this section, we summarize our work and main contributions in each chapter.

In Chapter 2, we implemented a CIP approach to TSP Race, where the goal is to minimize the time passing between receiving the graph and finishing the travel. Different from the traditional TSP, we also include the computation time in the objective function. To minimize the completion time, we used four CIP policies to parallelize computation with travel. Parallelization can increase the tour length, since

we are making more myopic decisions; however, almost all the computation can be performed while traveling. We showed that using the right setting it is possible to attain faster overall completion time. To determine how the parallelization should be conducted, we made a breakeven analysis for one of the CIP policies using regression. We tested the breakeven estimator on many different instances, and showed that it is very accurate to decide when to use CIP or the conventional compute-first-implement-later approach. As an extension to this chapter, we made a geometric analysis of the 2*opt* heuristic. We derived an expression of a region where there are no improvement moves between certain edges.

In Chapter 3, we addressed the computational challenges in routing problems where computation and loading require significant time. We identified CIP policies to parallelize the computation for routing with the loading phase. We implemented our CIP policies on Toth and Vigo's [67] efficient Granular Tabu Search (GTS) algorithm. We proposed and tested CIP policies both for LIFO and random access loading trucks, and evaluated their performance under three time allocation rules. We showed that CIP enables us to reach high quality solutions by allocating less time for a separate computation, and embedding some of the computation into loading.

As an extension to this chapter, we implemented a modification to the GTS algorithm. The modification is quite simple and easy to implement; it only changes the criterion used to create the sparse graph. We showed that the modification produces better results 76% of the time and improves the solution quality by 2% on average on instances with uniformly randomly distributed customers.

Our research in Chapter 2 led to a question on TSP tour length estimation models. There are many tour length estimators in the literature, but none of them has the following three properties at the same time: (i) does not require constructing a tour (ii) accurate (estimates within 6% of the optimal) and (iii) can be used in all node dispersions. In Chapter 4, we developed a new tour length estimation model based on

regression which can be used independent of the node distribution. Empirically our model is shown to be favorable compared to the ones in the literature. We also showed that our estimation model reduces to Beardwood et al.'s [7] well-known estimation formula when the node distribution is known, which provides a further support for its accuracy.

5.2 Recommendations for Future Research

In this dissertation, we introduced a CIP approach for specific cases of TSP and VRP. Our findings so far are promising to implement CIP on other versions of these routing problems as well as other problems in time-sensitive applications. In the rest of this section, we list some research problems which can benefit from CIP implementation.

Cancer treatment: Computer-driven radiation therapy is composed of sequential steps. These are mainly obtaining the image of the region that will be treated, optimizing the specifics of the beams, e.g., number of beams, dose of radiation, etc., and performing the actual delivery. Implementing CIP on beam optimization would not be acceptable, since the quality of those decisions is critical. However, once the beam specifics are determined, CIP can be implemented on the computation for the actual delivery, and time passing between taking the image and finishing the delivery can be decreased. This can be a significant benefit because during the procedure patient can move and the initial image may not perfectly reflect the current position at the time of delivery, which could result in unintended suboptimality of solutions. The less time elapses from the start of computation to the end of treatment, the more likely the solution is to remain good.

Studying the effect of uncertainties: In CTL-CVLRP, some of the savings can be traded off to let the trucks leave the warehouse early. If there are frequent interruptions in the delivery schedule due to unexpected traffic, breakdowns, bad weather conditions, etc., their negative impact can be alleviated. In the TSP Race problem,

uncertainties in travel times can be incorporated into the CIP settings.

Implementing the tour length estimation model in a LRP heuristic: We have shown that our tour length estimation model can generate very accurate estimates on many node dispersions. However, we have not exploited its possible uses. Nagy and Salhi [59] showed how to use tour length estimates as approximate solutions in LRP solution methods. Since our model is independent of the node dispersion, using it in a LRP heuristic to estimate the tour length can improve both the speed and the accuracy of those solution methods.

It is very likely that our CIP approach can be applied to other problem settings and applications as well.

REFERENCES

[1] Applegate, D. L., Bixby, R. E., Chvatal, V., and Cook, W. J., *The traveling salesman problem: a computational study*. Princeton University Press, 2007.

[2] Ausiello, G., Demange, M., Laura, L., and Paschos, V., "Algorithms for the on-line quota traveling salesman problem," *Information Processing Letters*, vol. 92, no. 2, pp. 89–94, 2004.

[3] Balas, E., Ceria, S., and Cornuéjols, G., "A lift-and-project cutting plane algorithm for mixed 0–1 programs," *Mathematical programming*, vol. 58, no. 1-3, pp. 295–324, 1993.

[4] Baldacci, R., Toth, P., and Vigo, D., "Exact algorithms for routing problems under vehicle capacity constraints," *Annals of Operations Research*, vol. 175, no. 1, pp. 213–245, 2010.

[5] Barnhart, C. and Laporte, G., *Handbooks in Operations Research & Management Science: Transportation: Transportation*, vol. 14. Elsevier, 2006.

[6] Bartholdi III, J. J., Platzman, L. K., Collins, R. L., and Warden III, W. H., "A minimal technology routing system for meals on wheels," *Interfaces*, vol. 13, no. 3, pp. 1–8, 1983.

[7] Beardwood, J., Halton, J. H., and Hammersley, J. M., "The shortest path through many points," in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 55, pp. 299–327, Cambridge Univ Press, 1959.

[8] Bock, F., "An algorithm for solving traveling-salesman and related network optimization problems," in *Unpublished manuscript associated with talk presented at the 14th ORSA National Meeting*, 1958.

[9] Bortfeldt, A. and Homberger, J., "Packing first, routing seconda heuristic for the vehicle routing and loading problem," *Computers & Operations Research*, vol. 40, no. 3, pp. 873–885, 2013.

[10] Boudia, M., Louly, M. A. O., and Prins, C., "Fast heuristics for a combined production planning and vehicle routing problem," *Production Planning and Control*, vol. 19, no. 2, pp. 85–96, 2008.

[11] Bräysy, O. and Gendreau, M., "Vehicle routing problem with time windows, part i: Route construction and local search algorithms," *Transportation science*, vol. 39, no. 1, pp. 104–118, 2005.

[12] Bräysy, O. and Gendreau, M., "Vehicle routing problem with time windows, part ii: Metaheuristics," *Transportation science*, vol. 39, no. 1, pp. 119–139, 2005.

[13] Bzdusek, K., Friberger, H., Eriksson, K., Hårdemark, B., Robinson, D., and Kaus, M., "Development and evaluation of an efficient approach to volumetric arc therapy planning," *Medical physics*, vol. 36, p. 2328, 2009.

[14] Carpaneto, G., Dell'Amico, M., Fischetti, M., and Toth, P., "A branch and bound algorithm for the multiple depot vehicle scheduling problem," *Networks*, vol. 19, no. 5, pp. 531–548, 1989.

[15] Chandra, B., Karloff, H., and Tovey, C., "New results on the old k-opt algorithm for the traveling salesman problem," *SIAM Journal on Computing*, vol. 28, no. 6, pp. 1998–2029, 1999.

[16] Chen, Z.-L., "Integrated production and outbound distribution scheduling: review and extensions," *Operations Research*, vol. 58, no. 1, pp. 130–148, 2010.

[17] Chen, Z.-L. and Vairaktarakis, G. L., "Integrated scheduling of production and distribution operations," *Management Science*, vol. 51, no. 4, pp. 614–628, 2005.

[18] Chien, T. W., "Operational estimators for the length of a traveling salesman tour," *Computers & operations research*, vol. 19, no. 6, pp. 469–478, 1992.

[19] Chien, T. W., "Heuristic procedures for practical-sized uncapacitated location-capacitated routing problems*," *Decision Sciences*, vol. 24, no. 5, pp. 995–1021, 1993.

[20] Christofides, N. and Eilon, S., "Expected distances in distribution problems," *OR*, pp. 437–443, 1969.

[21] Clarke, G. u. and Wright, J., "Scheduling of vehicles from a central depot to a number of delivery points," *Operations research*, vol. 12, no. 4, pp. 568–581, 1964.

[22] Cook, W. J., *In Pursuit of the Traveling Salesman: Mathematics at the Limits of Computation*. Princeton University Press, 2011.

[23] Croes, G., "A method for solving traveling-salesman problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958.

[24] Cunha, C. B. and Mutarelli, F., "A spreadsheet-based optimization model for the integrated problem of producing and distributing a major weekly news-magazine," *European Journal of Operational Research*, vol. 176, no. 2, pp. 925–940, 2007.

[25] Daganzo, C. F., "The distance traveled to visit n points with a maximum of c stops per vehicle: An analytic model and an application," *Transportation Science*, vol. 18, no. 4, pp. 331–350, 1984.

[26] Daganzo, C. F., "The length of tours in zones of different shapes," *Transportation Research Part B: Methodological*, vol. 18, no. 2, pp. 135–145, 1984.

[27] Davis, R. I. and Burns, A., "A survey of hard real-time scheduling for multiprocessor systems," *ACM Computing Surveys (CSUR)*, vol. 43, no. 4, p. 35, 2011.

[28] del Castillo, J. M., "A heuristic for the traveling salesman problem based on a continuous approximation," *Transportation Research Part B: Methodological*, vol. 33, no. 2, pp. 123–152, 1998.

[29] Eilon, S., Watson-Gandy, C. D. T., and Christofides, N., *Distribution management: mathematical modelling and practical analysis*. Griffin London, 1971.

[30] Englert, M., Röglin, H., and Vöcking, B., "Worst case and probabilistic analysis of the 2-opt algorithm for the tsp," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1295–1304, Society for Industrial and Applied Mathematics, 2007.

[31] Freeman, H. and Shapira, R., "Determining the minimum-area encasing rectangle for an arbitrary closed curve," *Communications of the ACM*, vol. 18, no. 7, pp. 409–413, 1975.

[32] Fuellerer, G., Doerner, K. F., Hartl, R. F., and Iori, M., "Ant colony optimization for the two-dimensional loading vehicle routing problem," *Computers & Operations Research*, vol. 36, no. 3, pp. 655–673, 2009.

[33] Fuellerer, G., Doerner, K. F., Hartl, R. F., and Iori, M., "Metaheuristics for vehicle routing problems with three-dimensional loading constraints," *European Journal of Operational Research*, vol. 201, no. 3, pp. 751–759, 2010.

[34] Gavish, B. and Srikanth, K., "An optimal solution method for large-scale multiple traveling salesmen problems," *Operations Research*, vol. 34, no. 5, pp. 698–717, 1986.

[35] Geismar, H. N., Laporte, G., Lei, L., and Sriskandarajah, C., "The integrated production and transportation scheduling problem for a product with a short lifespan," *INFORMS Journal on Computing*, vol. 20, no. 1, pp. 21–33, 2008.

[36] Gendreau, M., Iori, M., Laporte, G., and Martello, S., "A tabu search algorithm for a routing and container loading problem," *Transportation Science*, vol. 40, no. 3, pp. 342–350, 2006.

[37] Gendreau, M., Iori, M., Laporte, G., and Martello, S., "A tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints," *Networks*, vol. 51, no. 1, pp. 4–18, 2008.

[38] Gomory, R. E., "Outline of an algorithm for integer solutions to linear programs," *Bulletin of the American Mathematical Society*, vol. 64, no. 5, pp. 275–278, 1958.

[39] Halper, R. and Raghavan, S., "The mobile facility routing problem," *Transportation Science*, vol. 45, no. 3, pp. 413–434, 2011.

[40] Helbig Hansen, K. and Krarup, J., "Improvements of the heldkarp algorithm for the symmetric traveling-salesman problem," *Mathematical Programming*, vol. 7, no. 1, pp. 87–96, 1974.

[41] Held, M. and Karp, R. M., "The traveling-salesman problem and minimum spanning trees," *Operations Research*, vol. 18, no. 6, pp. 1138–1162, 1970.

[42] Held, M. and Karp, R. M., "The traveling-salesman problem and minimum spanning trees: Part ii," *Mathematical programming*, vol. 1, no. 1, pp. 6–25, 1971.

[43] http://www.iwr.uni heidelberg.de/groups/comopt/software/TSPLIB95/tsp/, March, 2013.

[44] http://www.math.uwaterloo.ca/tsp/concorde/index.html, November, 2013.

[45] http://www.math.uwaterloo.ca/tsp/world/countries.html, March, 2013.

[46] Iori, M., Salazar-González, J.-J., and Vigo, D., "An exact approach for the vehicle routing problem with two-dimensional loading constraints," *Transportation Science*, vol. 41, no. 2, pp. 253–264, 2007.

[47] Johnson, D., "Local optimization and the traveling salesman problem," *Automata, languages and programming*, pp. 446–461, 1990.

[48] Johnson, D. S. and McGeoch, L. A., "The traveling salesman problem: A case study in local optimization," *Local search in combinatorial optimization*, pp. 215–310, 1997.

[49] Kirkpatrick, S., Jr., D. G., and Vecchi, M. P., "Optimization by simmulated annealing," *science*, vol. 220, no. 4598, pp. 671–680, 1983.

[50] Kwon, O., Golden, B., and Wasil, E., "Estimating the length of the optimal tsp tour: an empirical study using regression and neural networks," *Computers & operations research*, vol. 22, no. 10, pp. 1039–1046, 1995.

[51] Kytöjoki, J., Nuortio, T., Bräysy, O., and Gendreau, M., "An efficient variable neighborhood search heuristic for very large scale vehicle routing problems," *Computers & Operations Research*, vol. 34, no. 9, pp. 2743–2757, 2007.

[52] Laporte, G., "Fifty years of vehicle routing," *Transportation Science*, vol. 43, no. 4, pp. 408–416, 2009.

[53] Lee, Y. H., Jung, J. W., and Lee, K. M., "Vehicle routing scheduling for cross-docking in the supply chain," *Computers & Industrial Engineering*, vol. 51, no. 2, pp. 247–256, 2006.

[54] Lin, S. and Kernighan, B. W., "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[55] Mester, D. and Bräysy, O., "Active guided evolution strategies for large-scale vehicle routing problems with time windows," *Computers & Operations Research*, vol. 32, no. 6, pp. 1593–1614, 2005.

[56] Nagata, Y., "Fast eax algorithm considering population diversity for traveling salesman problems," in *Evolutionary Computation in Combinatorial Optimization*, pp. 171–182, Springer, 2006.

[57] Nagata, Y., "Edge assembly crossover for the capacitated vehicle routing problem," in *Evolutionary Computation in Combinatorial Optimization*, pp. 142–153, Springer, 2007.

[58] Nagata, Y. and Bräysy, O., "Efficient local search limitation strategies for vehicle routing problems," in *Evolutionary Computation in Combinatorial Optimization*, pp. 48–60, Springer, 2008.

[59] Nagy, G. and Salhi, S., "Nested heuristic methods for the location-routeing problem," *Journal of the Operational Research Society*, pp. 1166–1174, 1996.

[60] Nagy, G. and Salhi, S., "Location-routing: Issues, models and methods," *European Journal of Operational Research*, vol. 177, no. 2, pp. 649–672, 2007.

[61] Otto, K., "Volumetric modulated arc therapy: Imrt in a single gantry arc," *Medical physics*, vol. 35, p. 310, 2008.

[62] Platzman, L. K. and Bartholdi III, J. J., "Spacefilling curves and the planar travelling salesman problem," *Journal of the ACM (JACM)*, vol. 36, no. 4, pp. 719–737, 1989.

[63] Robusté, F., Estrada, M., and López-Pita, A., "Formulas for estimating average distance traveled in vehicle routing problems in elliptic zones," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 1873, no. 1, pp. 64–69, 2004.

[64] Robusté, F., Daganzo, C. F., and Souleyrette II, R. R., "Implementing vehicle routing models,"

[65] Shepard, D., Cao, D., Afghan, M., and Earl, M., "An arc-sequencing algorithm for intensity modulated arc therapy," *Medical physics*, vol. 34, p. 464, 2007.

[66] Toth, P. and Vigo, D., *The vehicle routing problem*. Siam, 2001.

[67] Toth, P. and Vigo, D., "The granular tabu search and its application to the vehicle-routing problem," *INFORMS Journal on Computing*, vol. 15, no. 4, pp. 333–346, 2003.

[68] Ullrich, C. A., "Integrated machine scheduling and vehicle routing with time windows," *European Journal of Operational Research*, vol. 227, no. 1, pp. 152–165, 2013.

[69] Volgenant, T. and Jonker, R., "A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation," *European Journal of Operational Research*, vol. 9, no. 1, pp. 83–89, 1982.

[70] Zachariadis, E. E., Tarantilis, C. D., and Kiranoudis, C. T., "A guided tabu search for the vehicle routing problem with two-dimensional loading constraints," *European Journal of Operational Research*, vol. 195, no. 3, pp. 729–743, 2009.

[71] Zhang, P., Happersett, L., Hunt, M., Jackson, A., Zelefsky, M., and Mageras, G., "Volumetric modulated arc therapy: planning and evaluation for prostate cancer cases," *International Journal of Radiation Oncology* Biology* Physics*, vol. 76, no. 5, pp. 1456–1462, 2010.